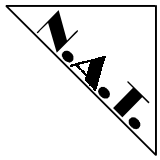
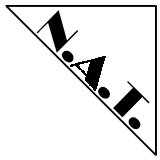




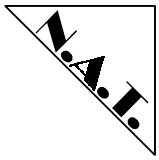

# TCP/IP – Socket Library
# Programmer's Reference
# Version 1.6




**N.A.T. GmbH**
**Kamillenweg 22**
**D-53757 Sankt Augustin**

**Phone: ++49/2241/3989-0**
**Fax: ++49/2241/3989-10**

**E-Mail: support@nateurope.com**
**Internet: http://www.nateurope.com**



# Disclaimer

The following documentation, compiled by N.A.T. GmbH (henceforth called N.A.T.), represents the current status of the products development. The documentation is updated on a regular basis. Any changes which might ensue, including those necessitated by updated specifications, are considered in the latest version of this documentation. N.A.T. is under no obligation to notify any person, organisation, or institution of such changes or to make these changes public in any other way.

We must caution you, that this publication could include technical inaccuracies or typographical errors.

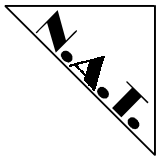
N.A.T. offers no warranty, either expressed or implied, for the contents of this documentation or for the product described therein, including but not limited to the warranties of merchantability or the fitness of the product for any specific purpose.

In no event, will N.A.T. be liable for any loss of data or for errors in data utilisation or processing resulting from the use of this product or the documentation. In particular, N.A.T. will not be responsible for any direct or indirect damages (including lost profits, lost savings, delays or interruptions in the flow of business activities, including but not limited to, special, incidental, consequential, or other similar damages) arising out of the use of or inability to use this product or the associated documentation, even if N.A.T. or any authorised N.A.T. representative has been advised of the possibility of such damages.

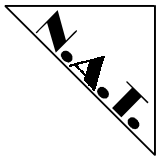
The use of registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations (patent laws, trade mark laws, etc.) and therefore free for general use. In no case does N.A.T. guarantee that the information given in this documentation is free of such third-party rights.



This product (and the associated documentation) is governed by the N.A.T. General Conditions and Terms of Delivery and Payment.



# Table of Contents



# 1 Introduction

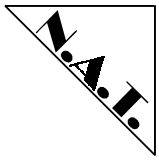
This manual provides a description of th eprogrammer's interface to the N.A.T. TCP/IP and N.A.T. OSI/TP protocol software. This includes the socket interface, conversion and address manipulating routines and function concerning the network data basis.
The interface can be used on both the TCP/IP and the ISO/OSI protocol stack. There are only slight differences as to the implementation on these different protocols. In case the usage of a specific function should be different on TCP/IP and ISO/OSI, the differences will be explained; any other functions can be used in the same way on both protocols.

# 2 Socket Interface

The following chapter presents in detail the individual functions of the socket interface in detail. These functions may be found in the library socklib.1. The individual functions are:

| | |
|---|---|
| **accept()** | Accept a request for a socket and create it. |
| **bind()** | Bind a name to a socket. |
| **close()** | Close a connection to a socket |
| **connect()** | Initiate a connection to a socket |
| **gethostname()** | Return the name of your own host system |
| **getpeername()** | Return the socket names of the connected systems |
| **getsockname()** | Return the name of your own socket |
| **getsockopt()** | View the values of a socket´s options |
| **ioctl()** | Manipulate the device parameters |
| **listen()** | Take requests for a connection to a socket |
| **read()** | Read data from a socket |
| **recv()** | Read data from a socket |
| **recvfrom()** | Read data from a socket |
| **recvmsg()** | Read data from a socket |
| **select()** | Simultaneously supervise multiple I/O paths |
| **send()** | Write data to a socket |
| **sendx()** | extended send() call (only on FDDI29/ETH29) |
| **sendmsg()** | Write data to a socket |
| **sendto()** | Write data to a socket |
| **setsockopt()** | Set socket options |
| **shutdown()** | Shutdown a portion of a full-duplex connection |
| **socket()** | Create a communications end point – socket |
| **SysReqMem()** | allocate system memory (only on FDDI29/ETH29) |
| **SysRetMem()** | return system memory (only on FDDI29/ETH29) |
| **write()** | Write data to a socket |

The functions gethostname(), ioctl(), read(), select(), sendx(), SysReqMem(), SysRetMem() and write() are extensions to the standard BSD socket interface.



## 2.1 General difference between TCP/IP and ISO/OSI

Between the implementations of the socket interface on TCP/IP and ISO/OSI there is one difference which is of general nature: wherever the struct sockaddr structure appears in the description of the socket interface in this manual (as it is used with TCP/IP) it has to be replaced by the struct sockaddr_iso structure when working with the OSI socket interface. This has the simple reason that the syntax of OSI addresses and TCP/IP addresses is defferent

The struct sockaddr structure used on TCP/IP is defined as follows:

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

This is the struct in_addr structure:

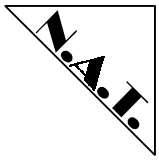
```
struct in_addr {
        u_long s_addr;
};
```

The following is the struct sockaddr_iso structure used on ISO/OSI:

```
struct sockaddr_iso {
        u_char siso_len;
        u_char siso_family;
        u_char siso_plen;
        u_char siso_slen;
        u_char siso_tlen;
        struct iso_addr siso_addr;
        u_char siso_pad[6];
};
```

This is the struct iso_addr structure:

```
struct iso_addr {
        u_char isoa_len;          /* length (in bytes) */
        char isoa_genaddr[20];    /* general opaque address */
};
```



## 2.2 accept()

**NAME**

accept() - accept a connection on a socket

**SYNOPSIS**

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

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

**DESCRIPTION**

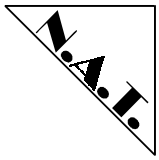
The argument s is a socket that has been created with socket(), bound to an address with bind(), and is listening for connections after a listen(). Accept extracts the first connection on the queue of pending connections, creates a new socket with the same properties of s and allocates a new file descriptor, ns, for the socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, accept blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket, ns, may not be used to accept more connections. The original socket s remains open.

The argument addr is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the domain in which the communication is occurring. The addrlen is a value-result parameter; it should initially contain the amount of space pointed to by addr; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types, currently with SOCK_STREAM for TCP/IP or SOCK_SEQPACKET for ISO/OSI.

It is possible to select() a socket for the purposes of doing an accept() by selecting it for read().

**RETURN VALUE**

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



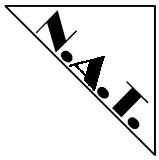
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | The descriptor is invalid. |
| **[EINVAL]** | A listen() must be executed before the accept(). |
| **[EWOULDBLOCK]** | The socket s is marked non-blocking and no connections are present to be accepted. |
| **[ECONNABORTED]** | The socket s is cannot receive any more data. |
| **[EOPNOTSUPP]** | The referenced socket is not of type SOCK_STREAM or SOCK_SEQPACKET. |
| **[ENOBUFS]** | There are not enough system resources to execute the function. |

**SEE ALSO**

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



## 2.3 bind()

**NAME**

bind() - bind a name to a socket

**SYNOPSIS**

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

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

**DESCRIPTION**

Bind assigns a name to an unnamed socket. When a socket is created with socket() it exists in a name space (address family) but has no name assigned. Bind requests that name be assigned to the socket s.

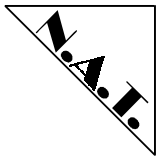
**RETURN VALUE**

If the bind is successful, a 0 value is returned. A return value of -1 indicates an error, which is further specified in the global errno.

**ERRORS**

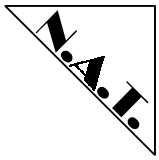
The call fails if:

| | |
|---|---|
| **[EBADF]** | S is not a valid path number for a socket descriptor. |
| **[EADDRNOTAVAIL]** | The specified address (name) is not available from the local machine. |
| **[EADDRINUSE]** | The specified address (name) is already in use. |
| **[E_ILLARG]** | The specified length namelen does not match the length of names in the specified address family. |
| **[EACCES]** | The requested address is protected, and the current user has inadequate permission to access it. |
| **[ENOBUFS]** | There are not enough system resources to execute the function. |




**SEE ALSO**

connect(), listen(), socket(), getsockname()



## 2.4 close()

**NAME**

close() - delete a descriptor

**SYNOPSIS**

*close(s)*

*int s;*

**DESCRIPTION**

The close call corresponds to the normal OS-9 close()- function and is used in exactly the same manner. The parameter s is a socket that has been created using either socket() or accept(). When a socket is no longer in use, close is used to break the connection and free all the system resources which were used by this socket.

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and the global integer variable errno is set to indicate the error.

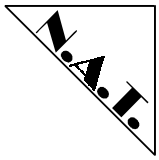
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | s is not an active descriptor. |

**SEE ALSO**

accept(), socket()



## 2.5 connect()

**NAME**

connect() - initiate a connection on a socket

**SYNOPSIS**

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

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

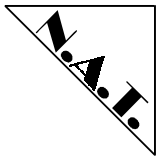
**DESCRIPTION**

The parameter s is a socket. If it is of type SOCK_DGRAM, then this call specifies the peer with which the socket is to be associated; this address is that to which datagrams are to be sent, and the only address from which datagrams are to be received. If the socket is of type SOCK_STREAM (or SOCK_SEQPACKET on ISO/OSI), then this call attempts to make a connection to another socket. The other socket is specified by name, which is an address in the communications space of the socket.

Each communications space interprets the name parameter in its own way. Generally, stream sockets may successfully connect only once; datagram sockets may use connect multiple times to change their association. Datagram sockets may dissolve the association by connecting to an invalid address, such as a null address.

**RETURN VALUE**

If the connection or binding succeeds, then 0 is returned. Otherwise a -1 is returned, and a more specific error code is stored in errno.



**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | S is not a valid descriptor. |
| **[EADDRINUSE]** | The address is already in use. |
| **[EADDRNOTAVAIL]** | The specified address is not available on this machine. |
| **[EAFNOSUPPORT]** | Addresses in the specified address family cannot be used with this socket. |
| **[EALREADY]** | The socket is non-blocking and a previous connection attempt has not yet been completed. |
| **[ECONNREFUSED]** | The attempt to connect was forcefully rejected. |
| **[EINPROGRESS]** | The socket is non-blocking and the connection cannot be completed immediately. It is possible to select() for completion by selecting the socket for writing. |
| **[EISCONN]** | The socket is already connected. |
| **[ENETUNREACH]** | The network isn't reachable from this host. |
| **[ENOBUFS]** | There are not enough system resources available to execute the function. |
| **[E_ILLARG]** | The specified namelen parameter specifies does not match the length of the names in the specified address family. |

**SEE ALSO**

accept(), select(), socket(), getsockname()

## 2.6 gethostname()

**NAME**

gethostname(), sethostname() - get/set name of current host

**SYNOPSIS**

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

*int sethostname(name, namelen)*
*char *name;*
*int namelen;*

**DESCRIPTION**

Gethostname() returns the standard host name for your own processor. The parameter namelen specifies the size of the name array. The returned name is null-terminated unless insufficient space is provided.
Gethostname() searches for the name by first checking the assigned location in the sock-descriptor and then (assuming that there was no string found) in reading the file /DD/ETC/systemid. If an entry for the host name is not found in either location, an ENOHNAME error is returned. If an entry for the name is found, the string is copied to the parameter name.
The host name is normally automatically written in the file /DD/ETC/systemid during the software configuration by the utility natconfig. If the file does not already exist natconfig will create it. Choosing natconfig´s sequence of menu selections Configuration and then Software will result in a dialog requesting the name of your own host processor. (It is also possible use an ASCII text editor to make the entry in the file /DD/ETC/systemid.) The file /DD/ETC/systemid should consist of just the host name and a terminating carriage return <CR>.

If you wish to enter the name in the sock-descriptor, you must use an editor to make the entry on the line labeled HNAME. The descriptor must then be generated again using make. The descriptor´s source and make files are located in the subdirectory SOURCE in the main TCPIP directory.

**RETURN VALUE**

If the call succeeds a value of 0 is returned. If the call fails, then a value of -1 is returned and an error code is placed in the global location errno.

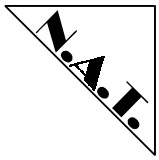
**ERRORS**

The call fails if:

| | |
|---|---|
| **[ENOHNAME]** | No host name was found in either the sock descriptor or the file /DD/ETC/systemid. |

**NOTE**

Host names are limited to a length of 64 characters.



## 2.7 getpeername()

**NAME**

getpeername() - get name of connected peer

**SYNOPSIS**

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

**DESCRIPTION**

Getpeername returns the name of the peer connected to socket s. The namelen parameter should be initialized to indicate the amount of space pointed to by name. On return, it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

**RETURN VALUE**

A 0 is returned if the call succeeds, -1 if it fails and an appropriate error code is returned in errno.

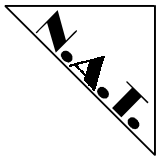
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | The argument s is not a valid descriptor. |
| **[ENOTCONN]** | The socket is not connected. |
| **[ENOBUFS]** | Insufficient resources were available in the system to perform the operation. |

**SEE ALSO**

accept(), bind(), socket(), getsockname()



## 2.8 getsockname()

**NAME**

getsockname() - get socket name

**SYNOPSIS**

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

**DESCRIPTION**

Getsockname returns the current name for the specified socket. The namelen parameter should be initialized to indicate the amount of space pointed to by name. On return it contains the actual size of the name returned (in bytes). The name will be truncated, if the specified buffer is too small.

**RETURN VALUE**

A 0 is returned if the call succeeds, -1 if it fails and an appropriate error code is returned in errno.

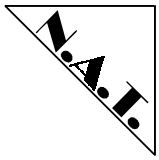
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | The argument s is not a valid descriptor. |
| **[ENOBUFS]** | Insufficient resources were available in the system to perform the operation. |

**SEE ALSO**

bind(), socket()



### 2.9 getsockopt()

**NAME**

getsockopt() - get socket options

**SYNOPSIS**

*#include <natdefs.h>*
*#include <socket.h>*

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

**DESCRIPTION**

Getsockopt can be used to view the options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost ``socket" level.

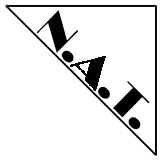
When viewing socket options, the level at which the option resides and the name of the option must be specified. To view options at the ``socket" level, level is specified as SOL_SOCKET. To view options at any other level, the protocol number of the appropriate protocol controlling the option must be entered. For example, to indicate that an option is to be interpreted by the TCP protocol, level should be set to the protocol number of TCP; see getprotoent().

For getsockopt(), the parameters optval and optlen identify a buffer in which the value for the requested option() are to be returned. For getsockopt, optlen is a value-result parameter, initially containing the size of the buffer pointed to by optval, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, optval may be supplied as 0.

Optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation.
The include file <socket.h> contains definitions for ``socket" level options, described below. Options at other protocol levels vary in format and name; consult the corresponding entries in the appropriate manuals.

# Manual difference:

Most socket-level options take an int parameter for optval. For setsockopt, the parameter should non-zero to enable a boolean option, or zero if the option is to be disabled. SO_LINGER uses a linger structure parameter, defined in <socket.h>, which



specifies the desired state of the option and the linger interval (see below).

The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt and set with setsockopt.

SO_DEBUG toggle recording of debugging information
SO_REUSEADDR toggle local address reuse
SO_KEEPALIVE toggle keep connections alive
SO_DONTROUTE toggle routing bypass for outgoing messages
SO_LINGER linger on close if data present
SO_BROADCAST toggle permission to transmit broadcast messages
SO_OOBINLINE toggle reception of out-of-band data in band
SO_SNDBUF set buffer size for output
SO_RCVBUF set buffer size for input
SO_TYPE get the type of the socket (get only)
SO_ERROR get and clear error on the socket (get only)

SO_DEBUG enables debugging in the underlying protocol modules. The data is written to specific address areas and can later be viewed using the program netprint.

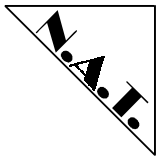
SO_REUSEADDR indicates that the rules used in validating addresses supplied in a bind() call should allow reuse of local addresses.
SO_KEEPALIVE enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal.

SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messages are queued on socket and a close() is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the process on the close attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the setsockopt call when SO_LINGER is requested). If SO_LINGER is disabled and a close is issued, the system will process the close in a manner that allows the process to continue as quickly as possible.

SO_BROADCAST requests permission to send broadcast datagrams on the socket. Broadcast packets are only used to send data to groups of recipients.

SO_OOBINLINE requests that out-of-band data (with protocols that support out-of-band data) be placed in the normal data input queue as received; it will then be accessible with recv() or read calls without the MSG_OOB flag.



SO_SNDBUF and SO_RCVBUF are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values.

Finally, SO_TYPE and SO_ERROR are options used only with getsockopt.

SO_TYPE returns the type of the socket, such as SOCK_STREAM; it is useful for servers that inherit sockets on startup.

SO_ERROR returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

**RETURN VALUE**

A 0 is returned if the call succeeds, if it fails a -1 is returned and a corresponding error code is written in the global variable errno.

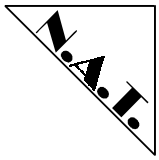
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | The argument s is not a valid descriptor. |
| **[ENOPROTOOPT]** | The option is unknown at the level indicated. |

**SEE ALSO**

bind(), close(), getprotobyname(), getprotobynumber(), getprotoent(), ioctl(), recv(), setsockopt(), socket()



## 2.10 ioctl()

**NAME**

ioctl() - set or read device characteristics

**SYNOPSIS**

*#include <ioctl.h>*

*int ioctl(d, request, argp)*
*int d;*
*unsigned long request;*
*char *argp;*

**DESCRIPTION**

Ioctl performs a variety of functions on open descriptors. In particular, many operating characteristics of character special files (e.g. terminals) may be controlled with ioctl requests. The write-up of various devices in the manual pages discuss how ioctl applies to them.

An ioctl request has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument argp in bytes. Macros and defines used in specifying an ioctl request are located in the file <ioctl.h>.

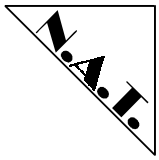
**RETURN VALUE**

A 0 is returned if the call succeeds, if it fails a -1 is returned and a corresponding error code is written in the global variable errno.

**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | D is not a valid descriptor. |



## 2.11 listen()

**NAME**

listen() - listen for connections on a socket

**SYNOPSIS**

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

**DESCRIPTION**

To accept connections, a socket is first created with socket(), a willingness to accept incoming connections and a queue limit for incoming connections are specified with listen(), and then the connections are accepted with accept(). The listen call applies only to sockets of type SOCK_STREAM or SOCK_SEQPACKET.

The backlog parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full, the client may receive an error with an indication of ECONNREFUSED, or, if the underlying protocol supports retransmission, the request may be ignored so that retries may succeed.

**RETURN VALUE**

A 0 is returned if the call succeeds, if it fails a -1 is returned and a corresponding error code is written in the global variable errno.

**ERRORS**

The call fails if:

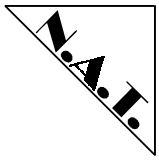
| | |
|---|---|
| **[EBADF]** | The argument s is not a valid descriptor. |
| **[EOPNOTSUPP]** | The socket is not of a type that supports the operation listen. |

**SEE ALSO**

accept(), connect(), socket()

**NOTE**

The backlog is currently limited (silently) to 5.



## 2.12 read()

**NAME**

read() - read input from a socket

**SYNOPSIS**

*int read(socket, buf, len)*
*int socket;*
*char *buf;*
*int len;*

**DESCRIPTION**

Read is an equivalent for the normal OS-9 read() - function, and is used in exactly the same manner. It attempts to read len bytes of data from the object referenced by the descriptor socket into the buffer pointed to by buf.

Normally, if no data is ready to be read from the socket, read() waits until data can be read. However, if the socket is marked as non-blocking (see ioctl()), read will not wait - instead it will return a -1 and write the value EWOULDBLOCK in the global variable errno.

The functions _gs_rdy() or select() can be used to determine whether or not data is available to be read at the socket.

**RETURN VALUE**

If successful, the number of bytes actually read is returned. Otherwise, a -1 is returned and the global variable errno is set to indicate the error.

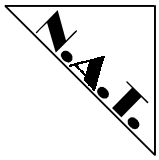
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | D is not a valid file or socket descriptor open for reading. |
| **[EWOULDBLOCK]** | The file was marked for non-blocking I/O, and no data were ready to be read. |

**SEE ALSO**

_gs_rdy(), accept(), ioctl(), recv(), select(), socket(), write()



## 2.13 recv()

**NAME**

recv() - receive a message from a socket

**SYNOPSIS**

*#include <natdefs.h>*
*#include <socket.h>*

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

**DESCRIPTION**

recv() is used to receive messages from a socket. The recv() call is normally used only on a connected socket (see connect()).
Socket defines the socket from which the data should be read. The parameter buf is the address of the buffer into which the received data will be written and len is the length of this buffer.

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is non-blocking (see ioctl()) in which case a return code of -1 is returned with the global variable errno set to EWOULDBLOCK.

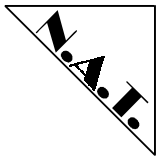
The _gs_rdy() and select() calls may be used to determine when more data arrives.

The flags argument to a recv call is formed by OR'ing one or more of the following values,

```
#define MSG_OOB         0x1   /* process out-of-band data */
#define MSG_PEEK              0x2   /* peek at incoming message */
```

**RETURN VALUE**

recv() returns the number of bytes successfully received, or, if an error occurred, a return code of -1 is returned with the global variable errno set to the appropriate error code.



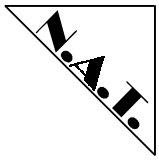
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | The argument s is an invalid descriptor. |
| **[EWOULDBLOCK]** | The socket is marked non-blocking and the receive operation would block. |

**SEE ALSO**

_gs_rdy(), getsockopt(), ioctl(), read(), recvfrom(), recvmsg(), select(), send(), sendmsg(), sendto(), socket()



## 2.14 recvfrom()

**NAME**

recvfrom() - read data from a socket

**SYNOPSIS**

*#include <natdefs.h>*
*#include <socket.h>*

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

**DESCRIPTION**

recvfrom() is used to receive messages from a socket. Recvfrom may be used to receive data from a socket whether it is in a connected state or not (see connect()).

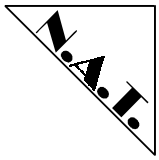
If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see socket()).
Socket defines the socket from which the data should be read. The parameter buf is the address of the buffer into which the received data will be written and len is the length of this buffer. If from is non-zero, the source address of the message is filled in. Fromlen is a value-result parameter, initialized to the size of the buffer associated with from, and modified on return to indicate the actual size of the address stored there.

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is non-blocking (see ioctl()) in which case a return code of -1 is returned with the global variable errno set to EWOULDBLOCK.

The _gs_rdy() and select() calls may be used to determine when more data arrives.

The flags argument to a recv call is formed by OR'ing one or more of the following values:

```
#define MSG_OOB         0x1    /* process out-of-band data */
#define MSG_PEEK              0x2    /* peek at incoming message */
```



**RETURN VALUE**

recvfrom() returns the number of bytes successfully received, or, if an error occurred, a return code of -1 is returned with the global variable errno set to the appropriate error code.

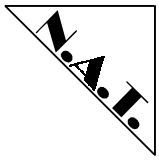
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | The argument s is an invalid descriptor. |
| **[EWOULDBLOCK]** | The socket is marked non-blocking and the receive operation would block. |

**SEE ALSO**

_gs_rdy(), getsockopt(), ioctl(), read(), recvfrom(), recvmsg(), select(), send(), sendmsg(), sendto(), socket()



## 2.15 recvmsg()

**NAME**

recvmsg() - read data from a socket

**SYNOPSIS**

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

int  recvmsg(socket, msg, flags)
        int socket;
        struct msghdr *msg;
        int flags;
```

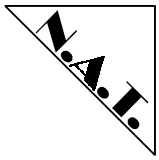
**DESCRIPTION**

recvmsg() is used to receive messages from a socket. recvmsg() may be used to receive data from a socket whether it is in a connected state or not (see connect()).

Socket is the socket from which the data should be read. The parameter msg is a pointer to a msghdr structure that the recvmsg call uses to minimize the number of directly supplied parameters. This structure has the following form, as defined in <socket.h>:

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

Here msg_name and msg_namelen specify the destination address if the socket is unconnected; msg_name may be given as a null pointer if no names are desired or required. A buffer to receive any access rights sent along with the message is specified in msg_accrights, which has the length msg_accrightslen. The parameter msg_iov is a pointer to the array of iovec structures with msg_iovlen defining the number of structures in the array. The iovec structures is used hold the addresses to which the received files should be copied. The iovec structure is defined in the uio.h file and has the following form:



```
struct iovec {
        caddr_t         io_base;
        int             iov_len;
};
```

Where:

**iov_base** is the start address for a memory area which may be used to write the received data.

**iov_len** is the available length of this memory area in bytes

Each memory area defined in the array is completely filled before the pointer msg_iov steps to the next area. Each memory area is used in the order of its appearance in the array.

If no messages are available at the socket, recvmsg() waits for a message to arrive, unless the socket is nonblocking (see ioctl()) in which case a return code of -1 is returned with the external variable errno set to EWOULDBLOCK.

The _gs_rdy() or select() calls may be used to determine when more data has arrived at the socket.

The flags argument to a recvmsg() call is formed by OR'ing one or more of the values,

```
#define MSG_OOB         0x1   /* process out-of-band data */
#define MSG_PEEK        0x2   /* peek at incoming message */
```

**RETURN VALUE**

recvmsg() returns the number of bytes successfully received, or, if an error occurred, a return code of -1 is returned with the global variable errno set to the appropriate error code.

**ERRORS**

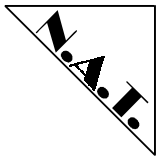
The calls fail if:

**[EBADF]** The argument s is an invalid descriptor.

**[EWOULDBLOCK]** The socket is marked non-blocking and the receive operation would block.

**SEE ALSO**

getsockopt(), ioctl(), read(), recvfrom(), recvmsg(), select(), send(), sendmsg(), sendto(), setsockopt(), socket()



## 2.16 select

**NAME**

select() - synchronous I/O path controlling

**SYNOPSIS**

*#include <natdefs.h>*
*#include <time_stuff.h>*

*int select(nfds, readfds, writefds, exceptfds, timeout)*
*int nfds;*
*fd_set *readfds, *writefds, *exceptfds;*
*struct timeval *timeout;*

*FD_SET(fd, &fdset)*
*FD_CLR(fd, &fdset)*
*FD_ISSET(fd, &fdset)*
*FD_ZERO(&fdset)*
*int fd;*
*fd_set fdset;*

**DESCRIPTION**

Select examines the I/O descriptor sets whose addresses are passed in readfds, writefds, and exceptfds to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The first nfds descriptors are checked in each set; i.e. the descriptors from 0 through nfds-1 in the descriptor sets are examined. On return, select replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned in nfound.

The descriptor sets are stored as bit fields in arrays of integers. The following macros (defined in natdefs.h) are provided for manipulating such descriptor sets:

| | |
|---|---|
| **FD_ZERO(&fdset)** | zeros the descriptor set fdset |
| **FD_SET(fd, &fdset)** | sets the bit for the descriptor fd in fdset. |
| **FD_CLR(fd, &fdset)** | deletes the bit for the descriptor fd from fdset. |
| **FD_ISSET(fd, &fdset)** | is nonzero if fd is a member of fdset, zero otherwise. |

The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to FD_SETSIZE, which is normally at least equal to the maximum number of descriptors supported by the system.

If timeout is a non-zero pointer, it specifies a maximum interval to wait for the selection to complete. If timeout is a zero pointer, the select blocks indefinitely. To affect a poll, the timeout argument should be non-zero, pointing to a zero-valued timeval structure.

Any of readfds, writefds, and exceptfds may be given as zero pointers if no descriptors are of interest.

**RETURN VALUE**

Select returns the number of ready descriptors that are contained in the descriptor sets, or -1 if an error occurred. If the time limit expires then select returns 0. If select returns with an error, including one due to an interrupted call, the descriptor sets will be unmodified.

**ERRORS**

The call fails if:

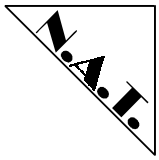
| | |
|---|---|
| **[EBADF]** | One of the descriptor sets specified an invalid descriptor. |
| **[EINTR]** | A signal was delivered before the time limit expired and before any of the selected events occurred. |
| **[EINVAL]** | The specified time limit is invalid. One of its components is negative or too large. |

**NOTE**

It is possible to select I/O paths which are not sockets. This is limited to those paths where _gs_rdy() may also be used. Such non-socket I/O paths may only be selected for reading.

**SEE ALSO**

accept(), connect(), _gs_rdy(), ioctl(), read(), recv(), recvfrom(), recvmsg(), select(), send(), sendmsg(), sendto(), socket(), write()



## 2.17 send()

**NAME**

send() - send a message from a socket

**SYNOPSIS**

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

int send(socket, buf, len, flags)
        int socket;
        char *buf;
        int len, flags;
```

**DESCRIPTION**

send() is used to transmit a message to another socket. send() may be used only when the socket is in a connected state.

Socket defines the socket from which the data should be read. The parameter buf is the address of the buffer from which the to be transmitted data will be read and len is the length of this buffer.

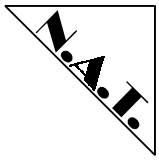
If no message space is available at the socket to hold the message to be transmitted, send() will normally block. However, if the socket has been placed in non-blocking I/O mode, send() will return a -1 and write the value EWOULDBLOCK in the global variable errno. The select() call may be used to determine when it is possible to send more data.

The flags parameter may include one or more of the following:

```
#define MSG_OOB          0x1    /* process out-of-band data */
#define MSG_DONTROUTE           0x4    /* bypass routing, use direct interface */
```

**MSG_OOB** is used to send out-of-band data on sockets that support this notion (e.g. SOCK_STREAM); the underlying protocol must also support out-of-band data.

**MSG_DONTROUTE** is usually used only by diagnostic or routing programs.




**RETURN VALUE**

The call returns the number of characters successfully sent, or, if an error occurred it will return a -1 and write the approriate error code in the global variable errno.

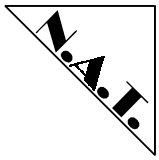
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | An invalid descriptor was specified. |
| **[EMSGSIZE]** | The socket requires that message be sent atomically, and the size of the message to be sent made this impossible. |
| **[EWOULDBLOCK]** | The socket is marked non-blocking and the requested operation would block. |

**SEE ALSO**

getsockopt(), ioctl(), recv(), recvfrom(), recvmsg(), select(), sendmsg(), sendto(), setsockopt(), socket(), write()



## 2.18 sendx()

**NAME**

sendx() – send a message from a socket (available only on ETH29/FDDI29)

**SYNOPSIS**

*int sendx(socket, sbuf, lbuf, length, flags)*
*int socket;*
*u_char *sbuf, *lbuf;*
*in length, flags;*

**DESRIPTION**

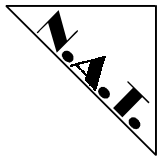
The sendx() call works in the same manner as the send() call, with one important difference: the address of the sendbuf is local to the FDDI29/ETH29's memory and not local to the host's memory in case of the send() call. Buffer space is allocated using the SysReqMem() call, it is returned with the SysRetMem() call.

Flags could be:

| | |
|---|---|
| **MSG_OOB** | 0x01 |
| **MSG_DONTROUTE** | 0x04 |
| **MSG_SWAPWORD** | 0x8000 |
| **MSG_SWAPBYTE** | 0x4000 |
| **MSG_USEDMA** | 0x2000 |

**SEE ALSO**

send(), SysReqMem(), SysRetMem()



## 2.19 sendmsg()

**NAME**

sendmsg() - send a message from a socket

**SYNOPSIS**

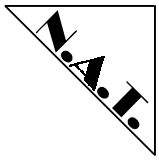
```
#include <natdefs.h>
#include <socket.h>

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

**DESCRIPTION**

sendmsg() is used to transmit a message to another socket. Sendmsg() may be used at any time regardless of whether the socket is connected or not (see connect()).
Socket defines the socket from which the data should be read. The parameter msg is a pointer to the msghdr structure into which the received data will be written. The parameter msg is used to reduce the number parameters to be passed. The structure msghdr is defined in the socket.h file and has the following form:

```
struct msghdr {
        caddr_t     msg_name;           /* optional address */
        int         msg_namelen;        /* length of the optional address */
        struct      iovec *msg_iov;     /* scatter/gather array */
        int         msg_iovlen;         /* # elements in msg_iov */
        caddr_t     msg_accrights;      /* access rights */
        int         msg_accrightslen;
};
```

Here msg_name and msg_namelen specify the destination address if the socket is unconnected; msg_name may be given as a null pointer if no names are desired or required. A buffer to receive any access rights sent along with the message is specified in msg_accrights, which has the length msg_accrightslen. The parameter msg_iov is a pointer to the array of iovec structures with msg_iovlen defining the number of structures in the array. The iovec structures is used hold the addresses from which the to be transmitted files should be copied. The iovec structure is defined in the uio.h file and has the following form:



```
struct iovec {
        caddr_t     io_base;
        int         iov_len;
};
```

Where:

**iov_base** is the start address for a memory area which may be used to read the to be transmitted data.

**iov_len** is the available length of this memory area in bytes

Each memory area defined in the array is completely read before the pointer msg_iov steps to the next area. Each memory area is read in the order of its appearance in the array.

If not enough message space is available at the socket to hold the message to be transmitted, then sendmsg() normally blocks, unless the socket has been placed in non-blocking I/O mode. The select() call may be used to determine when it is possible to send more data.

The flags parameter may include one or more of the following:

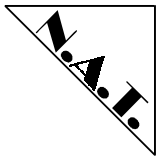
```
#define MSG_OOB          0x1    /* process out-of-band data */
#define MSG_DONTROUTE           0x4    /* bypass routing, use direct interface */
```

**MSG_OOB** is used to send out-of-band data on sockets that support this notion (e.g. SOCK_STREAM); the underlying protocol must also support out-of-band data.

**MSG_DONTROUTE** is usually used only by diagnostic or routing programs.

**RETURN VALUE**

The call returns the number of characters successfully sent, or, if an error occurred it will return a -1 and write the approriate error code in the global variable errno.



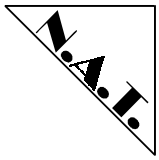
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | An invalid descriptor was specified. |
| **[EMSGSIZE]** | The socket requires that message be sent atomically, and the size of the message to be sent made this impossible. |
| **[EWOULDBLOCK]** | The socket is marked non-blocking and the requested operation would block. |

**SEE ALSO**

getsockopt(), ioctl(), recv(), recvfrom(), recvmsg(), select(), send(), sendto(), setsockopt(), socket(), write()



## 2.20 sendto()

**NAME**

sendto() - send a message from a socket

**SYNOPSIS**

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

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

**DESCRIPTION**

sendto() is used to transmit a message to another socket. sendto() may be used at any time regardless of whether a socket is connected or not (see connect()).

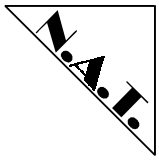
Socket defines the socket from which the data should be read. The parameter buf is the address of the buffer from which the data will be transmitted and len is the length of this buffer. The socket address of the target is given by to with tolen specifying its size. If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.

If no message space is available at the socket to hold the message to be transmitted, then sendto() normally blocks, unless the socket has been placed in non-blocking I/O mode. The select() call may be used to determine when it is possible to send more data.

The flags parameter may include one or more of the following:

```
#define MSG_OOB          0x1    /* process out-of-band data */
#define MSG_DONTROUTE           0x4    /* bypass routing, use direct interface */
```

| | |
|---|---|
| **MSG_OOB** | is used to send out-of-band data on sockets that support this notion (e.g. SOCK_STREAM); the underlying protocol must also support out-of-band data. |
| **MSG_DONTROUTE** | is usually used only by diagnostic or routing programs. |




**RETURN VALUE**

The call returns the number of characters successfully sent, or, if an error occurred it will return a -1 and write the approriate error code in the global variable errno.

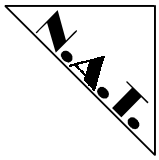
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | An invalid descriptor was specified. |
| **[EMSGSIZE]** | The socket requires that message be sent atomically, and the size of the message to be sent made this impossible. |
| **[EWOULDBLOCK]** | The socket is marked non-blocking and the requested operation would block. |

**SEE ALSO**

getsockopt(), ioctl(), recv(), recvfrom(), recvmsg(), select(), send(), sendmsg(), setsockopt(), socket(), write()



## 2.21 shutdown()

**NAME**

shutdown() - shut down part of a full-duplex connection

**SYNOPSIS**

*int shutdown(socket, how)*
*int socket, how;*

**DESCRIPTION**

shutdown() causes all or part of a full-duplex connection on the socket associated with socket to be shut down.
If how is 0, then further receives will be disallowed. If how is 1, then further sends will be disallowed. If how is 2, then further sends and receives will be disallowed. A send or receive function that attempts to access the socket after it has been shutdown will return an error code.

**RETURN VALUE**

A 0 is returned if the call succeeds, -1 if it fails and a corresponding error code is written to the global variable errno.

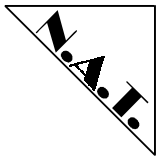
**ERRORS**

The call fails if:

| | |
|---|---|
| **[ENOTCONN]** | The specified socket is not connected. |

**SEE ALSO**

connect(), socket()



## 2.22 setsockopt()

**NAME**

setsockopt() - set socket options

**SYNOPSIS**

*#include <natdefs.h>*
*#include <socket.h>*

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

**DESCRIPTION**

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

When setting socket options, the level at which the option resides and the name of the option must be specified. To set options at the ``socket" level, level is specified as SOL_SOCKET. To manipulate options at any other level the protocol number of the appropriate protocol controlling the option must be entered. For example, to indicate that an option is to be interpreted by the TCP protocol, level should be set to the protocol number of TCP; see getprotoent().

The parameters optval and optlen are used by setsockopt to pass option values. If no option value is to be set, optval may be given as 0.

Optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file <socket.h> contains definitions for ``socket" level options, described below. Options at other protocol levels vary in format and name; consult the corresponding entries in the appropriate manuals.

### 2.22.1 Socket level options

Most socket-level options take an int parameter for optval. For setsockopt, the parameter should non-zero to enable a boolean option, or zero if the option is to be disabled. SO_LINGER uses a pointer to a linger structure parameter that is defined in <socket.h>, and which specifies the desired state of the option and the linger interval (see below).

The following options are recognized at the socket level. Except as noted, each may be examined with getsockopt and set with setsockopt.

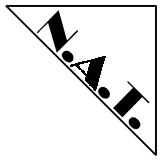
| | |
|---|---|
| **SO_DEBUG** | toggle recording of debugging information |
| **SO_REUSEADDR** | toggle local address reuse |
| **SO_KEEPALIVE** | toggle keep connections alive |
| **SO_DONTROUTE** | toggle routing bypass for outgoing messages |
| **SO_LINGER** | linger on close if data present |
| **SO_BROADCAST** | toggle permission to transmit broadcast messages |
| **SO_OOBINLINE** | toggle reception of out-of-band data in band |
| **SO_SNDBUF** | set buffer size for output |
| **SO_RCVBUF** | set buffer size for input |

**SO_DEBUG** enables debugging in the underlying protocol modules. The data is written to specific address areas and can later be viewed using the program netprint.

**SO_REUSEADDR** indicates that the rules used in validating addresses supplied in a bind() call should allow reuse of local addresses

**SO_KEEPALIVE** enables the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal.

**SO_DONTROUTE** indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.



| | |
|---|---|
| **SO_LINGER** | controls the action taken when unsent messages are queued on socket and a close() is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the process on the close attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the setsockopt call when SO_LINGER is requested). If SO_LINGER is disabled and a close is issued, the system will process the close in a manner that allows the process to continue as quickly as possible. |
| **SO_BROADCAST** | requests permission to send broadcast datagrams on the socket. Broadcast packets are only used to send data to groups of recipients. |
| **SO_OOBINLINE** | requests that out-of-band data (with protocols that support out-of-band data) be placed in the normal data input queue as received; it will then be accessible with recv() or read calls without the MSG_OOB flag. |
| **SO_SNDBUF and SO_RCVBUF** | are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values. |

### 2.22.2 IP multicasting

The IP multicasting capability has been implementad according to the RFC 1112. Currently it is available only for the ETH29 and FDDI29 boards. Of course, IP multicasting is not possible on ISO/OSI.

To enable IP multicasting, the parameter level has to be set to the protocol number of IP which is IPPROTO_IP.

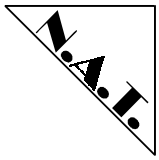
There are five options which concern the IP multicasting feature; they are passed in the optname parameter. Here is a detailed list of these options

**IP_MULTICAST_IF** Enables the multicasting capability on a given interface. The IP address of this interface is passed in the optval parameter which is a pointer to a struct in_addr structure (defined in the <netinet/in.h> include file). IP multicasting is disabled when the node has left the last multicasting group.

**IP_ADD_MEMBERSHIP** Adds a membership of this host to a specified multicasting group. The own IP address and the multicasting address of the group to be joined are specified in a struct ip_mreq structure, a pointer to which is passed in the optval parameter. The struct ip_mreq structure is defined in the <netinet/in.h> include file.

**IP_DROP_MEMBERSHIP** Deletes the membership of this host from a specified multicasting group. As in the IP_ADD_MEMBERSHIP call, the group is specified in a struct ip_mreq structire, a pointer to which is passed in the optval parameter.

**IP_MULTICAST_TTL** Used to change the Time To Live value of IP multicasting packets. The Time To Live value specifies how many gateways a multicasting message may pass and thus how far it will get in the network topography. The default Time To Live value is 1, so it has to be explicitely set to a higher value if multicast messages are supposed to reach hosts beyond the own network. The optval parameter is a pointer to an u:char containing the desired Time To Live value.



| | |
|---|---|
| **IP_MULTICAST_LOOP** | This option is used to specify whether or not IP multicasting messages are sent to the loopback interface. By default they are sent to the loopback interface since the sending host is itself a member of the corr. multicast group. The optval parameter is a pointer to an u_char containing TRUE (send to loopback interface) of FALSE (do not send to lopback interface). |

**RETURN VALUE**

A 0 is returned if the call succeeds, -1 if it fails and a corresponding error code is written to the global variable errno.

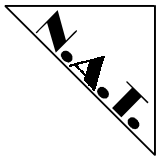
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | The argument s is not a valid descriptor. |
| **[ENOPROTOOPT]** | The option is unknown at the level given. |

**SEE ALSO**

bind(), close(), getprotobyname(), getprotobynumber(), getprotoent(3N), getsockopt(), ioctl(), recv(), socket(),



## 2.23 socket()

**NAME**

socket() - create an endpoint for communication

**SYNOPSIS**

*#include <natdefs.h>*
*#include <socket.h>*

*int socket(domain, type, protocol)*
*int domain, type, protocol;*

**DESCRIPTION**

socket() creates an endpoint for communication and returns an integer as a descriptor for it.

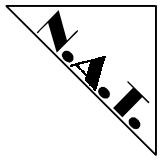
The domain parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file <socket.h>. The currently understood format is

**AF_INET** (ARPA Internet protocols)
**AF_ISO** (ISO/OSI)

The socket has the indicated type, which specifies the semantics of communication. The currently defined types are:

| | |
|---|---|
| **SOCK_STREAM** | type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. |
| **SOCK_DGRAM** | socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). |
| **SOCK_RAW** | sockets provide access to internal network protocols and interfaces. The type SOCK_RAW, which is available only to the super-user, is not described here. |
| **SOCK_SEQPACKET** | sequenced packet protocol; this must be used instead of SOCK_STREAM when working on ISO/OSI. |

The protocol specifies which protocol is to be used with the socket. Normally only a



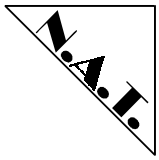
single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the communication domain in which communication is to take place; see protocols(3N).

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a connect() call. Once connected, data may be transferred using read() and write() calls or some variant of the send() and recv() calls. When a session has been completed a close() may be performed. Out-of-band data may also be transmitted as described in send() and received as described in recv().

The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with -1 returns and with ETIMEDOUT as the specific code in the global variable errno.

SOCK_DGRAM and SOCK_RAW sockets allow sending of datagrams to correspondents named in sendto() or sendmsg() calls. The send() and write() functions can be used with these types of sockets, if the destination is first set using connect(). Datagrams are generally received with recvfrom(), which returns the next datagram along with its return address.

The type SOCK_STREAM which is used in the TCP/IP stack for connection oriented sockets must be replaced by the SOCK_SEQPACKET type /sequenced packet protocol) when working on ISO/OSI.



The function ioctl() supports the command FIONBIO which can be used to toggle the socket between blocking and non-blocking mode.

```
#include <ioctl.h>
...
int s;
int nonblock = 1;
...
s = socket(AF_INET, SOCK_STREAM, 0);
...
if (ioctl(s, FIOBIO, (unsigned char *)&nonblock) <0) {
        perror("ioctl (FIOBIO)");
        exit(1);
}
```

By setting the variable nonblock to 0 the socket can be toggled back to the default setting "blocking".

The operation of sockets is controlled by socket level options. These options are defined in the file <socket.h>. Setsockopt() and getsockopt() are used to set and get options, respectively.

**RETURN VALUE**

A -1 is returned if an error occurs and a corresponding error code is written to the global variable errno. Otherwise, the return value is a descriptor referencing the socket.

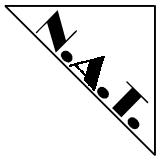
**ERRORS**

The call fails if:

| | |
|---|---|
| **[EPROTONOSUPPORT]** | The protocol type or the specified protocol is not supported within this domain. |
| **[EACCESS]** | Permission to create a socket of the specified type and/or protocol is denied. |
| **[ENOBUFS]** | Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed. |

**SEE ALSO**

accept(), bind(), connect(), getsockname(), getsockopt(), ioctl(), listen(), read(), recv(), recvfrom(), recvmsg(), select(), send(), sendmsg(), sendto(), setsockopt(), shutdown(), write()



## 2.24 SysReqMem()

**NAME**

SysReqMem() – allocate system memory on the FDDI29/ETH29 slave board

**SYNOPSIS**

*char *SysReqMem(socket, memsize)*
*int socket;*
*u_long memsize;*

**DESCRIPTION**

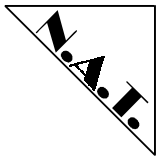
The SysReqMem() call is used to allocate system memory on the FDDI29/ETH29 slave board. The socket number supplied to the call is the one returned from a previous socket() call. The parameter memsize gives the requested memory size.

**RETURN VALUE**

Two return values are possible: 0, if no memory could be allocated in the desired size, or the address of the allocated memory.

**SEE ALSO**

send(), sendx(), SysRetMem()



## 2.25 SysRetMem()

**NAME**

SysRetMem() – retuirn system memory previously allocated by SysReqMem(9 on the FDDI29/ETH29 slave board

**SYNOPSIS**

*int SysRetMem(socket, memsize, memadr)*

*int socket;*
*u_long memsize;*
*char *memadr;*

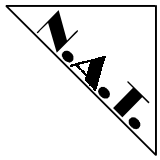
**DESCRIPTION**

The SysRetMem(9 call is used to return the system memory allocated previously by a SysReqMem() call. The parameters supplied to the call are the socket number and the memory size as already used by the SysReqMem() call, and the buffer address as returned by the SysReqMem() call.

SysRetMem() has to be called before a socket is closed. This is due to the fact that memory previously allocated by SysReqMem(9 is by no means related to any socket and therefore is not automatically freed when the socket is closed. But as the socket number is the only way to communicate with the slave board, you cannot free memory after you have closed the socket.

**SEE ALSO**

send(), sendx(), SysReqMem()



## 2.26 write()

**NAME**

write() - write output

**SYNOPSIS**

*int write(socket, buf, len)*
*int socket;*
*char *buf;*
*int len;*

**DESCRIPTION**

write() is an exact equivalent for the normal OS-9 write() - function and functions precisely the same. It attempts to write len bytes of data to the object referenced by the descriptor socket from the buffer pointed to by buf.
The parameter socket defines the socket from which the data should be read. The socket was created using either socket() or accept(). The parameter buf is the address of the buffer from which the to be transmitted data will be read and len is the length of this buffer.

If no message space is available at the socket to hold the message to be transmitted, write() will normally block. However, if the socket has been placed in non-blocking I/O mode, write() will return a -1 and write the value EWOULDBLOCK in the global variable errno. The select() call may be used to determine when it is possible to send more data.

If write() fails, the file pointer will remain unchanged.

**RETURN VALUE**

Upon successful completion, the number of bytes actually written is returned. Otherwise a -1 is returned and the global variable errno is set to indicate the error.

**ERRORS**

The call fails if:

| | |
|---|---|
| **[EBADF]** | Socket is not a valid descriptor open for writing. |
| **[EWOULDBLOCK]** | The Socket was marked for non-blocking I/O, and no data could be written immediately. |

**SEE ALSO**

accept(), ioctl(), read(), select(), send(), socket()

# 3 Conversion Routines

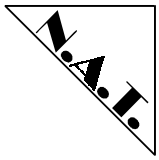
The following pages contain a description of the byte-order conversion utilities included in this package. These routines are provided as macros in the file in.h and can be used to convert words between the host and network ordering of bytes.

**htonl()** convert long word from host to network

**htons()** convert short word from host to network

**ntohl()** convert long word from network to host

**ntohs()** convert short word from network to host



### 3.1 htonl()

**NAME**

htonl() - Convert Long Word : Host -> Network

**DEFINITION**

*#include <in.h>*

*unsigned long htonl (hostlong)*
*unsigned long hostlong*

**DESCRIPTION**

The function htonl() converts a long word from the ordering used by the system where htonl() was called to the byte ordering required by the network.

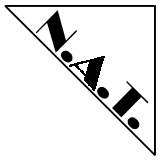
For systems with 680x0 processors, this function is available as a macro in the file in.h.

**RESULTS**

The word passed is returned in the network byte ordering.

SEE ALSO

gethostbyname(), getservent(), htons(), ntohl(), ntohs()



### 3.2 htons()

**NAME**

htons() - Convert Short Word : Host -> Network

**DEFINITION**

*#include <in.h>*

*unsigned short htons (hostshort)*
*unsigned short hostshort*

**DESCRIPTION**

The function htons() converts a short word from the ordering used by the system where htons() was called to the byte ordering required by the network.

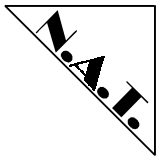
For systems with 680x0 processors, this function is available as a macro in the file in.h.

**RESULTS**

The word passed is returned in the network byte ordering.

**SEE ALSO**

gethostbyname(), getservent(), htons(), ntohl(), ntohs()



### 3.3 ntohl()

**NAME**

ntohl() - Convert Long Word : Network -> Host

**DEFINITION**

*#include <in.h>*

*unsigned long ntohl (netlong)*
*unsigned long netlong*

**DESCRIPTION**

The function ntohl() converts a long word from the ordering used by the network to the byte ordering required by the system where ntohl() was called.

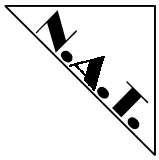
For systems with 680x0 processors, this function is available as a macro in the file in.h.

**RESULTS**

The word passed is returned in the byte ordering used internal to the system.

**SEE ALSO**

gethostbyname(), getservent(), htons(), ntohl(), ntohs()



### 3.4 ntohs()

**NAME**

ntohs() - Convert Short Word : Network -> Host

**DEFINITION**

*#include <in.h>*

*unsigned long ntohs (netshort)*
*unsigned long netshort*

**DESCRIPTION**

The function ntohs() converts a short word from the ordering used by the network to the byte ordering required by the system where ntohl() was called.

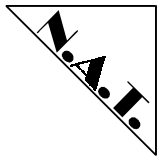
For systems with 680x0 processors, this function is available as a macro in the file in.h.

**RESULTS**

The word passed is returned in the byte ordering used internal to the system.

**SEE ALSO**

gethostbyname(), getservent(), htons(), ntohl(), ntohs()

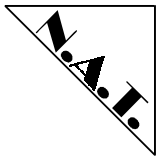


# 4 Address Manipulation Functions

The functions described in this chapter enable the manipulation of the Internet addresses. The functions can be found in the library inetdb.1.
Currently the ISO/OSI versions of these calls are not yet implemented. But in the near future, the address manipulating functions will be available for both TCP/IP and ISO/OSI, on ISO/OSI they begin with a preceding iso_ instead of inet_ than. It is necessary to differentiate in this way since the address formats used by both protocols are different, too. The functionality, however, is the same on both protocols.

These are the TCP/IP versions of the calls:

| | |
|---|---|
| **inet_addr()** | Generates a complete numerical Internet address from a string containing an Internet address in standard dot notation. |
| **inet_network()** | Generates a complete numerical Internet network number from a string containing an Internet address in standard dot notation. |
| **inet_ntoa()** | Generates a string in standard dot notation from a numerical Internet address. |
| **inet_makeaddr()** | Generates a complete numerical Internet address by combining a network number with a numerical host address. |
| **inet_lnaof()** | Extract the numerical host address from a numerical Internet address. |
| **inet_netof()** | Extract the network number from a numerical Internet address. |



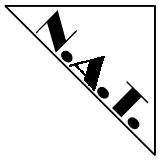
## 4.1 Internet Addresses in Dot Notation

Internet addresses represented in dot notation can take one of the following four forms:

| | |
|---|---|
| **a.b.c.d** | If four parts are given, ech part will be interpreted as a single byte. Each part (byte) will then be ordered from left to right to the four bytes of an Internet address. |
| **a.b.c** | If three parts are given, the right-most part (c) will be interpreted as a 16-bit unit and will be assigned to the two right-most bytes of the internet address. This agrees with the class B network address form 128.Net.Host. |
| **a.b** | If two parts are given, the last part (b) will be interpreted as a 24-bit unit and will be assigned to the 3 right-most bytes of the Internet address. This agrees with the class A network address form Net.Host. |
| **a** | If only a single part is given it will be treated as a direct entry of the Internet address. |

In dot notation all the parts of an Internet adress can be entered in decimal, octal or hexadecimal format. The syntax is the same as for the "C" programming language, thus:

- a leading 0x or 0X signifies that the next digit should be interpreted as **hexadecimal**,
- a leading 0 (zero) signifies that the next digit should be interpreted as **octal**, and
- all other cases should be treated as being in **decimal** notation.



## 4.2 inet_addr()

**NAME**

inet_addr() - Internet Address Dot -> Numerical

**DEFINITION**

*#include <socket.h>*
*#include <in.h>*
*#include <inet.h>*

*unsigned long inet_addr(cp)*
*char *cp;*

**DESCRIPTION**

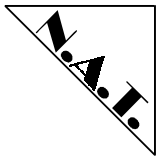
The function inet_addr() generates a complete numerical Internet address from a string containing an Internet address in standard dot notation. The parameter cp is a pointer to the Internet address string in dot notation. The bytes of the returned numerical Internet address are ordered as required by the network (from left to right).

**RESULTS**

Successful execution will return the numerical Internet address. If the Internet address in dot-notation was in the incorrect form, a -1 will be returned.

**SEE ALSO**

gethostbyname(), hosts(), inet_network(), inet_ntoa(), inet_makeaddr(), inet_lnaof(), inet_netof()



### 4.3 inet_lnaof()

**NAME**

inet_lnaof() - Extract Host Address from Numerical Internet Address

**DEFINITION**

*#include <socket.h>*
*#include <in.h>*
*#include <inet.h>*

*unsigned long inet_lnaof(in)*
*struct in_addr in;*

**DESCRIPTION**

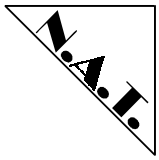
The function inet_lnaof() extracts the numerical host address from a numerical Internet address. The function returns the host address in the appropriate byte order for the system which called it.

**RESULTS**

Successful execution will return the numerical Internet host address. If the Internet address in dot-notation was in the incorrect form, a -1 will be returned.

**SEE ALSO**

gethostbyname(), hosts(), inet_addr(), inet_network(), inet_ntoa(), inet_makeaddr(), inet_netof()



### 4.4 inet_makeaddr()

**NAME**

inet_makeaddr - Network + Host Address -> Numerical Internet Address

**DEFINITION**

*#include <socket.h>*
*#include <in.h>*
*#include <inet.h>*

*struct in_addr inet_addr(net, lna)*
*int net, lna;*

**DESCRIPTION**

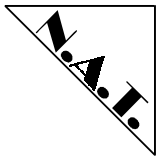
The function inet_makeaddr() generates a complete numerical Internet address by combining a network number with a numerical host address. The bytes of the returned numerical Internet address are ordered as required by the network (from left to right).

**RESULTS**

Successful execution will return the numerical Internet address in the structure in_addr (the 4 bytes are ordered from left to right).

**SEE ALSO**

gethostbyname(), hosts(), inet_network(), inet_ntoa(), inet_makeaddr(), inet_lnaof(), inet_netof()



### 4.5 inet_netof()

**NAME**

inet_netof() - Extract Network Address from Numerical Internet Address

**DEFINITION**

*#include <socket.h>*
*#include <in.h>*
*#include <inet.h>*

*unsigned long inet_netof(in)*
*struct in_addr in;*

**DESCRIPTION**

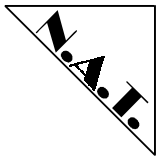
The function inet_netof() extracts the network number from a numerical Internet address. The bytes of the returned network number are ordered as required by the calling system.

**RESULTS**

Successful execution will return the network number. If the Internet address in dot-notation was in the incorrect form, a -1 will be returned.

**SEE ALSO**

gethostbyname(), hosts(), inet_addr(), inet_network(), inet_ntoa(), inet_makeaddr(), inet_lnaof()



### 4.6 inet_network()

**NAME**

inet_network() - Dot Network Address -> Numerical

**DEFINITION**

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

unsigned long inet_network(cp)
        char *cp;
```

**DESCRIPTION**

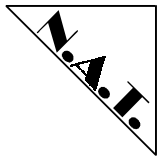
The function inet_network() generates a complete numerical Internet network number from a string containing an Internet address in standard dot notation. The parameter cp is a pointer to the string containing the Internet network address in dot notation. The bytes of the returned numerical Internet address are ordered as required by the calling system.

**RESULTS**

Successful execution will return the numerical network address. If the Internet network address in dot-notation was in the incorrect form, a -1 will be returned.

**SEE ALSO**

gethostbyname(), hosts(), inet_addr(), inet_ntoa(), inet_makeaddr(), inet_lnaof(), inet_netof()



### 4.7 inet_ntoa()

**NAME**

inet_ntoa() - Numerical Internet Address -> Dot

**DEFINITION**

*#include <socket.h>*
*#include <in.h>*
*#include <inet.h>*

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

**DESCRIPTION**

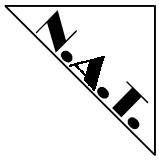
The function inet_ntoa( ) generates a string in standard dot notation from a numerical Internet address. The parameter cp is a pointer to the Internet address string in dot notation. The bytes of the returned numerical Internet address are ordered as required by the network (from left to right).

**RESULTS**

Successful execution will return the numerical Internet address. If the Internet address in dot-notation was in the incorrect form, a -1 will be returned.

**SEE ALSO**

gethostbyname(), hosts(), inet_addr(), inet_network(), inet_ntoa(), inet_makeaddr(), inet_lnaof(), inet_netof()



# 5 Network Data Basis

The following pages detail the files and functions used for accessing the network data basis. The functions can be found in the library netdb.1

The network data basis consists of the following files:

| | |
|---|---|
| **hosts** | data basis for host names |
| **networks** | data basis for network numbers |
| **protocols** | data basis for protocol names |
| **services** | data basis for service names |

The functions which can be used to view the data basis are:

To read host entries

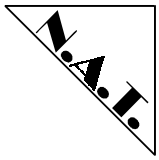
*gethostbyname()*
*gethostbyaddr()*
*gethostent()*
*sethostent()*
*endhostent()*

To read network entries

*getnetbyaddr()*
*getnetbyname()*
*getnetent()*
*setnetent()*
*endnetent()*

To read protocol entries

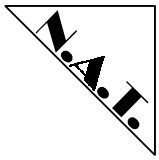
*getprotobyaddr()*
*getprotobyname()*
*getprotoent()*
*setprotoent()*
*endprotoent()*



To read services entries

*getservbyaddr()*
*getservbyname()*
*getservbyport()*
*getservent()*
*setservent()*
*endservent()*

So far, on ISO/OSI only the gethostbyname() call has been implemented.



## 5.1 hosts

**NAME**

hosts - host name data base

**DESCRIPTION**

The hosts file contains information regarding the known hosts on the network. For each host a single line should be present with the following information:

- official host name
- Internet address
- aliases

Items are separated by any number of blanks and/or tab characters. A ``#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Network addresses must be entered in the conventional Internet dot notation. Please refer to section 4.1, "Interner Addresses in Dot Notation".

Host names may contain any printable character other than a blank, newline, or comment character. If you wish to be connected into the worldwide Internet network, the hosts file should be based on the official host data base that is administered by the Network Information Control Center (NIC).

**FILES**

/DD/ETC/hosts

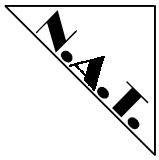
**EXAMPLE**

*# The official names are out of the greek, the further names out of the roman mythology*
*# The names and Internet addresses shown here are not from the official NIC host data base*

| | | | |
|---|---|---|---|
| *127.1* | *local* | *localhost lh* | |
| *192.1.1.1* | *zeus* | *jupiter* | |
| *192.1.1.2* | *hera* | *juno* | |
| *192.1.1.3* | *hermes* | *mercury* | |
| *192.1.1.4* | *odin* | *wotan* | *# in between a germanic* |
| *192.1.1.5* | *ares* | *mars* | *# back to greek/roman* |
| *192.1.1.6* | *aphrodite* | *venus* | *# the second "token woman"* |

**SEE ALSO**

ifconfig(), gethostbyaddr(), gethostbyname(), sethostent(), gethostent(), endhostent()



### 5.1.1 The struct hostent structure

**NAME**

struct hostent – the host entry structure

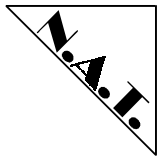
**DESCRIPTION**

The struct hostent structure is used by the functions concerned with the hosts database and describes an internet host. It is defined as follows:

```
struct hostent {
        char        *h_name;        /* official name of host */
        char        **h_aliases;    /* alias list */
        int         h_addrtype;     /* host address type */
        int         h_length;       /* length of address */
        char        **h_addr_list;  /* list of addresses from name server */
};
#define  h_addr h_addr_list[0]      /* address, for backward compatibility */
```

The members of this structure are:

| | |
|---|---|
| **h_name** | Official name of the host. |
| **h_aliases** | A zero terminated array of alternate names for the host. |
| **h_addrtype** | The type of address being returned; currentl always AF_INET. |
| **h_length** | The length, in bytes, of the address. |
| **h_addr_list** | A zero terminated array of network addresses for the host. Host addresses are returned in network byte order. |
| **h_addr** | The first address in h_addr_list; this is for backward compatibility. |



### 5.1.2 gethostbyaddr()

**NAME**

gethostbyaddr() - get network host entry

**SYNOPSIS**

*#include <natdefs.h>*
*#include <netdb.h>*

*extern int h_errno;*

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

**DESCRIPTION**

The function gethostbyaddr() takes the address of the desired host in the parameter addr, which has a length of len bytes and is of address type type (currently type can only be AF_INET).

**RESULTS**

When gehostbyaddr() has been executed successfully, it returns a pointer to a struct hostent structure describing the referenced host (see section 5.1.1, "The struct hostent structure").
Otherwise, an error return status from gethostbyaddr is indicated by return of a null pointer and the external variable h_errno will contain the associated error code.

**ERRORS**

The call fails if:

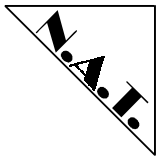
**HOST_NOT_FOUND** The specified host address is unknown.

**SEE ALSO**

gethostbyname(), sethostent(), gethostent(), endhostent(), hosts

**NOTE**

All information is contained in a static area so it must be copied, if it is to be saved. Only the Internet address format is currently understood.



### 5.1.3 gethostbyname()

**NAME**

gethostbyname() - get network host entry

**SYNOPSIS**

*#include <natdefs.h>*
*#include <netdb.h>*

*extern int h_errno;*
*struct hostent *gethostbyname(name)*
*char *name;*

**DESCRIPTION**

The function gethostbyname() takes the name of the desired host in the parameter name.

**DIFFERENCES ON ISO/OSI**

On acount of the mass of possible OSI address formats and the problems incident to that there currently is to standard which makes possible a homogeneous administration of addresses and host names, as in the case of TCP/IP with the file /dd/etc/hosts. The N.A.T. GmbH has developed a new gethostbyname() function for ISO/OSI in order to make the address administration as transparent as possible to the user of the N.A.T. OSI/TP protocol software; this function extracts the OSI-address of a given host name out of the /dd/etc/hosts file.

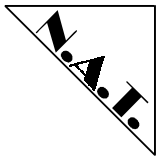
The gethostbyname() function call has a different systax for the protocol family AF_ISO which is responsible for OSI/TP:

*struct hostent *gethostbyname(name, addr_fam)*
*char *name;*
*int *addr_fam;*

where addr_fam has to be set to AF_ISO.

**RESULTS**

When gehostbyname() has been executed successfully, it returns a pointer to a struct hostent structure describing the referenced host (see section 5.1.1, "The struct hostent structure"). Otherwise, an error return status from gethostbyname is indicated by return of a null pointer and the external variable h_errno will contain the associated error code.



**ERRORS**

The call fails if:

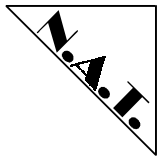
**HOST_NOT_FOUND** The specified host name is unknown.

**SEE ALSO**

gethostbyaddr(), sethostent(), gethostent(), endhostent(), hosts

**NOTE**

All information is contained in a static area so it must be copied if it is to be saved. Currently the AF_INET and AF_ISO address format are understood, but the syntax of the call is different for these formats.



### 5.1.4 gethostent()

**NAME**

gethostent() - get network host entry

**SYNOPSIS**

*#include <natdefs.h>*
*#include <netdb.h>*

*extern int h_errno;*
*struct hostent *gethostent()*

**DESCRIPTION**

The function gethostent() always gets the next entry in the hosts data base. The data base will be opened automatically if necessary.

**RESULTS**

When gehostent() has been executed successfully, it returns a pointer to a struct hostent structure (see section 5.1.1, "The struct hostent structure"). Otherwise, an error return status from gethostent() is indicated by return of a null pointer and the external variable h_errno will contain the associated error code.

**ERRORS**

The call fails if:

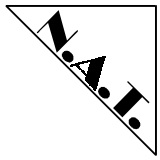
**HOST_NOT_FOUND** The specified host name is unknown.

**SEE ALSO**

gethostbyaddr(), gethostbyname(), sethostent(), endhostent(), hosts

**NOTE**

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.



### 5.1.5 sethostent()

**NAME**

sethostent() - open network hosts data base

**SYNOPSIS**

*#include <natdefs.h>*
*#include <netdb.h>*

*sethostent(stayopen)*
*int stayopen;*

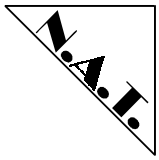
**DESCRIPTION**

The function sethostent() opens the hosts data base and positions an internal pointer to the first entry in this data base. A subsequent call to gethostent() will return a pointer to the hostent structure for the first entry in this data base (see section 5.1.1, "The struct hostent structure").

If the parameter stayopen contains a non-zero value, the hosts data base will remain open following subsequent calls to gethostbyaddr() or gethostbyname(). Normally, the hosts data base is closed again following every function call.

**SEE ALSO**

gethostbyaddr(), gethostbyname(), gethostent(), endhostent(), hosts



### 8.1.5 endhostent()

**NAME**

endhostent() - Close the hosts data base

**SYNOPSIS**

*#include <natdefs.h>*
*#include <netdb.h>*

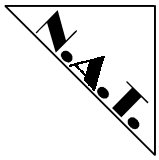
*endhostent()*

**DESCRIPTION**

The function endhostent() closes the hosts data base after use.

**SEE ALSO**

gethostbyaddr(), gethostbyname(), sethostent(), gethostent(), hosts



## 5.2 networks

**NAME**

networks - network name data base

**DESCRIPTION**

The networks file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

- official network name
- network number
- aliases

Items are separated by any number of blanks and/or tab characters. A ``#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional ``." notation using the inet_network() routine from the Internet address manipulation library, inet(). Network names may contain any printable character other than a field delimiter, newline, or comment character.

**EXAMPLE**

```
#
# Network entries
#
arpanet         10              arpa
matnet      128.14          # Mobile Access Terminal Net
aeronet         192.5.9         # Aerospace Labnet
olymp           192.1.1         # our own network
```

**FILES**

/DD/ETC/networks

**SEE ALSO**

ifconfig(), getnetbyaddr(), getnetbyname(), setnetent(), getnetent(), endnetent()

### 5.2.1 The struct netent structure

**NAME**

struct netent – the net entry structure

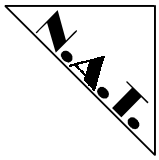
**DESCRIPTION**

The struct netent structure is used by the functions concerned with the network database and contains the broken-out fields of a line in the network data base, /DD/ETC/networks. it is defined as follows:

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

The members of this structure are:

| | |
|---|---|
| **n_name** | The official name of the network. |
| **n_aliases** | A zero terminated list of alternate names for the network. |
| **n_addrtype** | The type of the network number returned; currently only AF_INET. |
| **n_net** | The network number. Network numbers are returned in machine byte order. |



### 5.2.2 getnetbyaddr()

**NAME**

getnetbyaddr() - get network entry

**SYNOPSIS**

*#include <natdefs.h>*
*#include <netdb.h>*

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

**DESCRIPTION**

Getnetbyaddr() takes the parameters:

| | |
|---|---|
| **net** | for the network number of the desired network |
| **type** | for the address type (currently only AF_INET) |

**RESULTS**

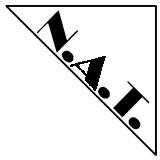
If getnetbyaddr() has been executed successfully, it returns a pointer to a strust netent structure (see section 5.2.1, "The struct netent structure"). Otherwise, an error return status from getnetbyaddr() is indicated by return of a null pointer.

**SEE ALSO**

getnetbyname(), setnetent(), getnetent(), endnetent(), networks

**NOTE**

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.



### 8.2.2 getnetbyname()

**NAME**

getnetbyname() - get network entry

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*struct netent *getnetbyname(name)*
*char *name;*

**DESCRIPTION**

Getnetbyname() takes the parameter:

| | |
|---|---|
| **name** | for the name of the desired network |

**RESULTS**

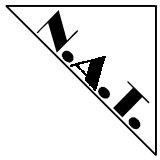
If getnetbyname() has been executed successfully, it returns a pointer to a struct netent structure (see section 5.2.1, "The struct netent structure"). Otherwise, an error return status from getnetbyname() is indicated by return of a null pointer.

**SEE ALSO**

getnetbyaddr(), setnetent(), getnetent(), endnetent(), networks

**NOTE**

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.



### 5.2.4 getnetent()

**NAME**

getnetent() - get network entry

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*struct netent *getnetent()*

**DESCRIPTION**

getnetent() always reads the next entry (line) of the networks data base, opening the file if necessary.

**RESULTS**

If getnetent() has been successfully executed, it returns a pointer to a struct netent structure (see section 5.2.1, "The struct netent structure"). Otherwise, an error return status from getnetent is indicated by return of a null pointer.

**SEE ALSO**

getnetbyaddr(), getnetbyname(), setnetent(), endnetent(), networks

**NOTE**

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.

### 8.2.4 setnetent()

**NAME**

setnetent() - Open the networks data base

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*setnetent(stayopen)*
*int stayopen;*

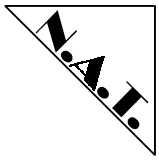
**DESCRIPTION**

The function setnetent() opens the networks data base and positions an internal pointer to the first entry in this data base. A subsequent call to getnetent() will return a pointer to the netent structure for the first entry in this data base (see section 5.2.1, "The struct netent structure").

If the parameter stayopen contains a non-zero value, the networks data base will remain open following subsequent calls to getnetbyaddr() or getnetbyname(). Normally, the networks data base is closed again following every function call.

**SEE ALSO**

getnetbyaddr(), getnetbyname(), getnetent(), endnetent(), networks



### 5.2.6 endnetent()

**NAME**

endnetent() - Close the networks data base

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

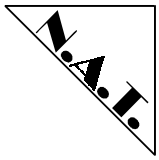
*endnetent()*

**DESCRIPTION**

The function endnetent() closes the networks data base file after use.

**SEE ALSO**

getnetaddr(), getnetbyname(), setnetent(), getnetent(), networks



## 8.3 protocols

**NAME**

protocols - protocol name data base

**DESCRIPTION**

The protocols file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

- official protocol name
- protocol number
- aliases

Items are separated by any number of blanks and/or tab characters. A ``#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

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

**EXAMPLE**

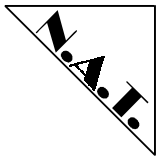
```
#
# Internet (IP) Protocols
#
ip          0       IP          # Internet Protocol
icmp        1       ICMP        # Internet Control Message Protocol
ggp         3       GGP         # Gateway-Gateway Protocol
tcp         6       TCP         # Transmission Control Protocol
egp         8       EGP         # Exterior Gateway Protocol
pup         12      PUP         # PARC Universal Packet Protocol
udp         17      UDP         # User Datagram Protocol
hmp         20      HMP         # Host Monitoring Protocol
xns-idp     22      XNS-IDP     # Xerox NS IDP
rdp         27      RDP         # "Reliable Datagram" Protocol
```

**FILES**

/DD/ETC/protocols

**SEE ALSO**

getprotobyname(), getprotobynumber(), getprotoent(), setprotoent(), endprotoent()



### 5.3.1 The struct protoent structure

**NAME**

struct protoent – the protocol entry structure

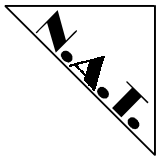
**DESCRIPTION**

The struct protoent structure is used by the funtions concerned with the network protocol data base and contains the broken-out fields of a line in the network protocol data base, /DD/ETC/protocols. It is defined as follows:

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

The members of this structure are:

| | |
|---|---|
| **p_name** | The official name of the protocol. |
| **p_aliases** | A zero terminated list of alternate names for the protocol. |
| **p_proto** | The protocol number. |



### 5.3.2 getprotobyname()

**NAME**

getprotobyname() - get protocol entry

**SYNOPSIS**

*#include <netdb.h>*

*struct protoent *getprotobyname(name)*
*char *name;*

**DESCRIPTION**

getprotobyname() takes the protocol name of the desired protocol in the parameter name.

**RESULTS**

If getprotobyname() has been successfully executed, it returns a pointer to a struct protoent structure (see section 5.3.1, "The struct protoent structure"). Otherwise, an error return status from getprotobyname() is indicated by return of a null pointer.

**SEE ALSO**

getprotobynumber(), getprotoent(), setprotoent(), endprotoent(), protocols

**NOTE**

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

### 5.3.3 getprotobynumber()

**NAME**

getprotobynumber() - get protocol entry

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*struct protoent *getprotobynumber(proto)*
*int proto;*

**DESCRIPTION**

getprotobynumber() takes the protocol number of the desired protocol in the parameter proto.

**RESULTS**

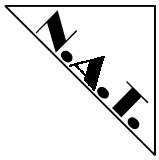
If getprotobynumber() has been successfully executed, it returns a pointer to a struct protoent structure (see section 5.3.1, "The struct protoent structure"). Otherwise, an error return status from getprotobynumber() is indicated by return of a null pointer.

**SEE ALSO**

getprotobyname(), getprotoent(), setprotoent(), endprotoent(), protocols

**NOTE**

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



### 5.3.4 getprotoent()

**NAME**

getprotoent() - get protocol entry

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*struct protoent *getprotoent()*

**DESCRIPTION**

The function getprotoent() reads the next line of the protocols data base, opening the file if necessary.

**RESULTS**

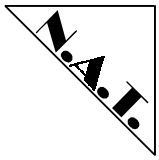
If getprotoent() has been successfully executed, it returns a pointer to a protoent structure (see section 5.3.1, "The struct protoent structure"). Otherwise, an error return status from getprotoent() is indicated by return of a null pointer.

**SEE ALSO**

getprotobyname(), getprotobynumber(), setprotoent(), endprotoent(), protocols

**NOTE**

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



### 5.3.5 setprotoent()

**NAME**

setprotoent() - Open the protocols data base

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*setprotoent(stayopen)*
*int stayopen*

**DESCRIPTION**

The function setprotoent() opens the protocols data base and positions an internal pointer to the first entry in this data base. A subsequent call to getprotoent() will return a pointer to the protoent structure for the first entry in this data base (see section 5.3.1, "The struct protoent structure").

If the parameter stayopen contains a non-zero value, the protocols data base will remain open following subsequent calls to getprotobyaddr() or getprotobyname(). Normally, the protocols data base is closed again following every function call.

**SEE ALSO**

getprotobyname(), getprotobynumber(), getprotoent(), endprotoent(), protocols

### 5.3.6 endprotoent()

**NAME**

endprotoent() - Close protocols data base

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

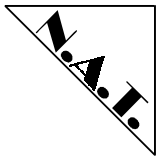
*endprotoent()*

**DESCRIPTION**

The function endprotoent() closes the network protocol data base, /DD/ETC/protocols.

**SEE ALSO**

getprotobyname(), getprotobynumber(), getprotoent(), setprotoent(), protocols



## 5.4 services

**NAME**

services - services name data base

**DESCRIPTION**

The services file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

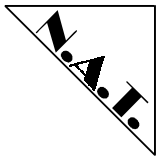
- official service name
- port number
- protocol name
- aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single item; a ``/" is used to separate the port and protocol (e.g. ``512/tcp"). A ``#" indicates the beginning of a comment; subsequent characters up to the end of the line are not interpreted by the routines which search the file.

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

**EXAMPLE**

```
#
# Internet Network Services
#
echo        7/tcp
echo        7/udp
discard         9/tcp       sink null
discard         9/udp       sink null
daytime     13/tcp
daytime     13/udp
netstat     15/tcp
chargen     19/tcp      ttytst source
chargen     19/udp      ttytst source
ftp         21/tcp
telnet      23/tcp
smtp        25/tcp mail
```

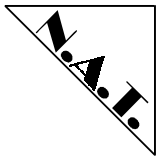


**FILES**

/DD/ETC/services

**SEE ALSO**

getservbyname(), getservbyport(), setservent(), getservent(), endservent()



### 5.4.1 The struct servent structure

**NAME**

struct servent – the service entry structure

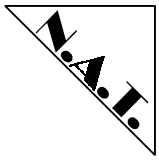
**DESCRIPTION**

The struct servent structure is used by the functions concerned with the network services data base and contains the broken-out fields of a line in the network services data base, /DD/ETC/services. It is defined as follows:

```
struct   servent {
        char *s_name;  /* official name of service */
        char **s_aliases;   /* alias list */
        int s_port;     /* port service resides at */
        char *s_proto; /* protocol to use */
};
```

The members of this structure are:

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



### 5.4.2 getservbyname()

**NAME**

getservbyname() - get service entry

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

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

**DESCRIPTION**

The function getservbyname() takes the port number of the desired service in the parameter port. A protocol name may be included for the service, however if it is it must agree with the port´s entry in the services data base.

**RESULTS**

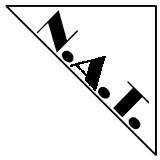
If getservbyname() has been successfully executed, it returns a pointer to a struct servent structure (see section 5.4.1, "The struct servent structure"). Otherwise, an error return status from getservbyname() is indicated by return of a null pointer.

**SEE ALSO**

getservbyport(), setservent(), getservent(), endservent(), services

**NOTE**

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



### 5.4.3 getservbyport()

**NAME**

getservbyport() - get service entry

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

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

**DESCRIPTION**

The function getservbyport() takes the port number of the desired service in the parameter port. A protocol name may be included for the service, however if it is it must agree with the port´s entry in the services data base.

**RESULTS**

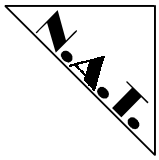
If getservbyport() has been successfully executed, it returns a pointer to a struct servent structure (see section 5.4.1, "The struct servent structure"). Otherwise, an error return status from getservbyport() is indicated by return of a null pointer.

**SEE ALSO**

getservbyname(), setservent(), getservent(), endservent(), services

**NOTE**

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



### 5.4.4 getservent()

**NAME**

getservent() - get service entry

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*struct servent *getservent()*

**DESCRIPTION**

The function getservent() reads the next entry/line of the services data base, opening the file if necessary.

**RESULTS**

If getservent() has been successfully executed, it returns a pointer to a struct servent structure (see section 5.4.1, "The struct servent structure"). Otherwise, an error return status from getservent() is indicated by return of a null pointer.

**SEE ALSO**

getservbyname(), getservbyport(), setservent(), endservent(), services

**NOTE**

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

### 5.4.5 setservent()

**NAME**

setservent() - Open services data base

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*setservent(stayopen)*
*int stayopen*

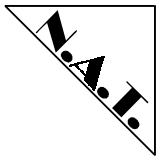
**DESCRIPTION**

The function setservent() opens the services data base and positions an internal pointer to the first entry in this data base. A subsequent call to getservent() will return a pointer to the struct servent structure for the first entry in this data base (see section 5.4.1, "The struct servent structure").

If the parameter stayopen contains a non-zero value, the services data base will remain open following subsequent calls to getservbyaddr() or getservbyname(). Normally, the services data base is closed again following every function call.

**SEE ALSO**

getservbyname(), getservbyport(), getservent(), endservent(), services



### 5.4.6 endservent()

**NAME**

endservent() - Close the services data base

**SYNOPSIS**

*#include <netdb.h>*
*#include <natdefs.h>*

*endservent()*

**DESCRIPTION**

The function endservent closes the network services data base, /DD/ETC/services.

**SEE ALSO**

getservbyname(), getservbyport(), setservent(), getservent(), services