# NAME

Socket, sockaddr_in, sockaddr_un, inet_aton, inet_ntoa, inet_pton, inet_ntop - load the C socket.h
defines and structure manipulators

# SYNOPSIS

```
use Socket;

$proto = getprotobyname('udp');
socket(Socket_Handle, PF_INET, SOCK_DGRAM, $proto);
$iaddr = gethostbyname('hishost.com');
$port = getservbyname('time', 'udp');
$sin = sockaddr_in($port, $iaddr);
send(Socket_Handle, 0, 0, $sin);

$proto = getprotobyname('tcp');
socket(Socket_Handle, PF_INET, SOCK_STREAM, $proto);
$port = getservbyname('smtp', 'tcp');
$sin = sockaddr_in($port,inet_aton("127.1"));
$sin = sockaddr_in(7,inet_aton("localhost"));
$sin = sockaddr_in(7,INADDR_LOOPBACK);
connect(Socket_Handle,$sin);

($port, $iaddr) = sockaddr_in(getpeername(Socket_Handle));
$peer_host = gethostbyaddr($iaddr, AF_INET);
$peer_addr = inet_ntoa($iaddr);

$proto = getprotobyname('tcp');
socket(Socket_Handle, PF_UNIX, SOCK_STREAM, $proto);
unlink('/var/run/usock');
$sun = sockaddr_un('/var/run/usock');
connect(Socket_Handle,$sun);
```

# DESCRIPTION

This module is just a translation of the C *socket.h* file. Unlike the old mechanism of requiring a
translated *socket.ph* file, this uses the **h2xs** program (see the Perl source distribution) and your native
C compiler. This means that it has a far more likely chance of getting the numbers right. This includes
all of the commonly used pound-defines like AF_INET, SOCK_STREAM, etc.

Also, some common socket "newline" constants are provided: the constants CR, LF, and CRLF, as
well as $CR, $LF, and $CRLF, which map to \015, \012, and \015\012. If you do not want to use
the literal characters in your programs, then use the constants provided here. They are not exported
by default, but can be imported individually, and with the :crlf export tag:

```
use Socket qw(:DEFAULT :crlf);
```

In addition, some structure manipulation functions are available:

inet_aton HOSTNAME

> Takes a string giving the name of a host, and translates that to an opaque string (if
> programming in C, struct in_addr). Takes arguments of both the 'rtfm.mit.edu' type and
> '18.181.0.24'. If the host name cannot be resolved, returns undef. For multi-homed hosts
> (hosts with more than one address), the first address found is returned.

> For portability do not assume that the result of inet_aton() is 32 bits wide, in other words, that
> it would contain only the IPv4 address in network order.

inet_ntoa IP_ADDRESS

> Takes a string (an opaque string as returned by inet_aton(), or a v-string representing the four octets of the IPv4 address in network order) and translates it into a string of the form 'd.d.d.d' where the 'd's are numbers less than 256 (the normal human-readable four dotted number notation for Internet addresses).

INADDR_ANY

> Note: does not return a number, but a packed string.

> Returns the 4-byte wildcard ip address which specifies any of the hosts ip addresses. (A particular machine can have more than one ip address, each address corresponding to a particular network interface. This wildcard address allows you to bind to all of them simultaneously.) Normally equivalent to inet_aton('0.0.0.0').

INADDR_BROADCAST

> Note: does not return a number, but a packed string.

> Returns the 4-byte 'this-lan' ip broadcast address. This can be useful for some protocols to solicit information from all servers on the same LAN cable. Normally equivalent to inet_aton('255.255.255.255').

INADDR_LOOPBACK

> Note - does not return a number.

> Returns the 4-byte loopback address. Normally equivalent to inet_aton('localhost').

INADDR_NONE

> Note - does not return a number.

> Returns the 4-byte 'invalid' ip address. Normally equivalent to inet_aton('255.255.255.255').

IN6ADDR_ANY

> Returns the 16-byte wildcard IPv6 address. Normally equivalent to inet_pton(AF_INET6, "::")

IN6ADDR_LOOPBACK

> Returns the 16-byte loopback IPv6 address. Normally equivalent to inet_pton(AF_INET6, "::1")

sockaddr_family SOCKADDR

> Takes a sockaddr structure (as returned by pack_sockaddr_in(), pack_sockaddr_un() or the perl builtin functions getsockname() and getpeername()) and returns the address family tag. It will match the constant AF_INET for a sockaddr_in and AF_UNIX for a sockaddr_un. It can be used to figure out what unpacker to use for a sockaddr of unknown type.

sockaddr_in PORT, ADDRESS

sockaddr_in SOCKADDR_IN

> In a list context, unpacks its SOCKADDR_IN argument and returns an array consisting of (PORT, ADDRESS). In a scalar context, packs its (PORT, ADDRESS) arguments as a SOCKADDR_IN and returns it. If this is confusing, use pack_sockaddr_in() and unpack_sockaddr_in() explicitly.

pack_sockaddr_in PORT, IP_ADDRESS

> Takes two arguments, a port number and an opaque string, IP_ADDRESS (as returned by inet_aton(), or a v-string). Returns the sockaddr_in structure with those arguments packed in with AF_INET filled in. For Internet domain sockets, this structure is normally what you need for the arguments in bind(), connect(), and send(), and is also returned by getpeername(), getsockname() and recv().

unpack_sockaddr_in SOCKADDR_IN

Takes a sockaddr_in structure (as returned by pack_sockaddr_in()) and returns an array of two elements: the port and an opaque string representing the IP address (you can use inet_ntoa() to convert the address to the four-dotted numeric format). Will croak if the structure does not have AF_INET in the right place.

sockaddr_in6 PORT, IP6_ADDRESS, [ SCOPE_ID, [ FLOWINFO ] ]

sockaddr_in6 SOCKADDR_IN6

In list context, unpacks its SOCKADDR_IN6 argument according to unpack_sockaddr_in6(). In scalar context, packs its arguments according to pack_sockaddr_in6().

pack_sockaddr_in6 PORT, IP6_ADDRESS, [ SCOPE_ID, [ FLOWINFO ] ]

Takes two to four arguments, a port number, an opaque string (as returned by inet_pton()), optionally a scope ID number, and optionally a flow label number. Returns the sockaddr_in6 structure with those arguments packed in with AF_INET6 filled in. IPv6 equivalent of pack_sockaddr_in().

unpack_sockaddr_in6 SOCKADDR_IN6

Takes a sockaddr_in6 structure (as returned by pack_sockaddr_in6()) and returns an array of four elements: the port number, an opaque string representing the IPv6 address, the scope ID, and the flow label. (You can use inet_ntop() to convert the address to the usual string format). Will croak if the structure does not have AF_INET6 in the right place.

sockaddr_un PATHNAME

sockaddr_un SOCKADDR_UN

In a list context, unpacks its SOCKADDR_UN argument and returns an array consisting of (PATHNAME). In a scalar context, packs its PATHNAME arguments as a SOCKADDR_UN and returns it. If this is confusing, use pack_sockaddr_un() and unpack_sockaddr_un() explicitly. These are only supported if your system has *<sys/un.h>*.

pack_sockaddr_un PATH

Takes one argument, a pathname. Returns the sockaddr_un structure with that path packed in with AF_UNIX filled in. For unix domain sockets, this structure is normally what you need for the arguments in bind(), connect(), and send(), and is also returned by getpeername(), getsockname() and recv().

unpack_sockaddr_un SOCKADDR_UN

Takes a sockaddr_un structure (as returned by pack_sockaddr_un()) and returns the pathname. Will croak if the structure does not have AF_UNIX in the right place.

inet_pton ADDRESS_FAMILY, HOSTNAME

Takes an address family, either AF_INET or AF_INET6, and a string giving the name of a host, and translates that to an opaque string (if programming in C, struct in_addr or struct in6_addr depending on the address family passed in). The host string may be a string hostname, such as 'www.perl.org', or an IP address. If using an IP address, the type of IP address must be consistent with the address family passed into the function.

This function is not exported by default.

inet_ntop ADDRESS_FAMILY, IP_ADDRESS

Takes an address family, either AF_INET or AF_INET6, and a string (an opaque string as returned by inet_aton() or inet_pton()) and translates it to an IPv4 or IPv6 address string.

This function is not exported by default.

getaddrinfo HOST, SERVICE, [ HINTS ]

Given at least one of a hostname and a service name, returns a list of address structures to listen on or connect to. HOST and SERVICE should be plain strings (or a numerical port

number for SERVICE). If present, HINTS should be a reference to a HASH, where the
following keys are recognised:

flags => INT

> A bitfield containing `AI_*` constants

family => INT

> Restrict to only generating addresses in this address family

socktype => INT

> Restrict to only generating addresses of this socket type

protocol => INT

> Restrict to only generating addresses for this protocol

The return value will be a list; the first value being an error indication, followed by a list of
address structures (if no error occured).

```
 my ( $err, @results ) = getaddrinfo( ... );
```

The error value will be a dualvar; comparable to the `EI_*` error constants, or printable as a
human-readable error message string. Each value in the results list will be a HASH reference
containing the following fields:

family => INT

> The address family (e.g. AF_INET)

socktype => INT

> The socket type (e.g. SOCK_STREAM)

protocol => INT

> The protocol (e.g. IPPROTO_TCP)

addr => STRING

> The address in a packed string (such as would be returned by
> pack_sockaddr_in)

canonname => STRING

> The canonical name for the host if the `AI_CANONNAME` flag was provided, or
> `undef` otherwise. This field will only be present on the first returned address.

getnameinfo ADDR, FLAGS

Given a packed socket address (such as from `getsockname`, `getpeername`, or returned by
`getaddrinfo` in a `addr` field), returns the hostname and symbolic service name it
represents. FLAGS may be a bitmask of `NI_*` constants, or defaults to 0 if unspecified.

The return value will be a list; the first value being an error condition, followed by the
hostname and service name.

```
 my ( $err, $host, $service ) = getnameinfo( ... );
```

The error value will be a dualvar; comparable to the `EI_*` error constants, or printable as a
human-readable error message string. The host and service names will be plain strings.