NAME

`Socket` - networking constants and support functions

SYNOPSIS

`Socket` a low-level module used by, among other things, the *IO::Socket* family of modules. The following examples demonstrate some low-level uses but a practical program would likely use the higher-level API provided by `IO::Socket` or similar instead.

```
use Socket qw(PF_INET SOCK_STREAM pack_sockaddr_in inet_aton);

socket(my $socket, PF_INET, SOCK_STREAM, 0)
    or die "socket: $!";

my $port = getservbyname "echo", "tcp";
connect($socket, pack_sockaddr_in($port, inet_aton("localhost")))
    or die "connect: $!";

print $socket "Hello, world!\n";
print <$socket>;
```

See also the *EXAMPLES* section.

DESCRIPTION

This module provides a variety of constants, structure manipulators and other functions related to socket-based networking. The values and functions provided are useful when used in conjunction with Perl core functions such as socket(), setsockopt() and bind(). It also provides several other support functions, mostly for dealing with conversions of network addresses between human-readable and native binary forms, and for hostname resolver operations.

Some constants and functions are exported by default by this module; but for backward-compatibility any recently-added symbols are not exported by default and must be requested explicitly. When an import list is provided to the `use Socket` line, the default exports are not automatically imported. It is therefore best practice to always to explicitly list all the symbols required.

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);

$sock->print("GET / HTTP/1.0$CRLF");
```

The entire getaddrinfo() subsystem can be exported using the tag `:addrinfo`; this exports the getaddrinfo() and getnameinfo() functions, and all the `AI_*`, `NI_*`, `NIx_*` and `EAI_*` constants.

CONSTANTS

In each of the following groups, there may be many more constants provided than just the ones given as examples in the section heading. If the heading ends `...` then this means there are likely more; the exact constants provided will depend on the OS and headers found at compile-time.

PF_INET, PF_INET6, PF_UNIX, ...

Protocol family constants to use as the first argument to socket() or the value of the `SO_DOMAIN` or `SO_FAMILY` socket option.

AF_INET, AF_INET6, AF_UNIX, ...

Address family constants used by the socket address structures, to pass to such functions as inet_pton() or getaddrinfo(), or are returned by such functions as sockaddr_family().

SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, ...

Socket type constants to use as the second argument to socket(), or the value of the `SO_TYPE` socket option.

SOCK_NONBLOCK. SOCK_CLOEXEC

Linux-specific shortcuts to specify the `O_NONBLOCK` and `FD_CLOEXEC` flags during a `socket(2)` call.

```
 socket( my $sockh, PF_INET, SOCK_DGRAM|SOCK_NONBLOCK, 0 )
```

SOL_SOCKET

Socket option level constant for setsockopt() and getsockopt().

SO_ACCEPTCONN, SO_BROADCAST, SO_ERROR, ...

Socket option name constants for setsockopt() and getsockopt() at the `SOL_SOCKET` level.

IP_OPTIONS, IP_TOS, IP_TTL, ...

Socket option name constants for IPv4 socket options at the `IPPROTO_IP` level.

IPTOS_LOWDELAY, IPTOS_THROUGHPUT, IPTOS_RELIABILITY, ...

Socket option value constants for `IP_TOS` socket option.

MSG_BCAST, MSG_OOB, MSG_TRUNC, ...

Message flag constants for send() and recv().

SHUT_RD, SHUT_RDWR, SHUT_WR

Direction constants for shutdown().

INADDR_ANY, INADDR_BROADCAST, INADDR_LOOPBACK, INADDR_NONE

Constants giving the special `AF_INET` addresses for wildcard, broadcast, local loopback, and invalid addresses.

Normally equivalent to inet_aton('0.0.0.0'), inet_aton('255.255.255.255'), inet_aton('localhost') and inet_aton('255.255.255.255') respectively.

IPPROTO_IP, IPPROTO_IPV6, IPPROTO_TCP, ...

IP protocol constants to use as the third argument to socket(), the level argument to getsockopt() or setsockopt(), or the value of the `SO_PROTOCOL` socket option.

TCP_CORK, TCP_KEEPALIVE, TCP_NODELAY, ...

Socket option name constants for TCP socket options at the `IPPROTO_TCP` level.

IN6ADDR_ANY, IN6ADDR_LOOPBACK

Constants giving the special `AF_INET6` addresses for wildcard and local loopback.

Normally equivalent to inet_pton(AF_INET6, "::") and inet_pton(AF_INET6, "::1") respectively.

IPV6_ADD_MEMBERSHIP, IPV6_MTU, IPV6_V6ONLY, ...

Socket option name constants for IPv6 socket options at the `IPPROTO_IPV6` level.

STRUCTURE MANIPULATORS

The following functions convert between lists of Perl values and packed binary strings representing structures.

$family = sockaddr_family $sockaddr

Takes a packed socket address (as returned by pack_sockaddr_in(), pack_sockaddr_un() or the perl builtin functions getsockname() and getpeername()). Returns the address family tag. This will be one of the `AF_*` constants, such as `AF_INET` for a `sockaddr_in` addresses or `AF_UNIX` for a `sockaddr_un`. It can be used to figure out what unpack to use for a sockaddr of unknown type.

$sockaddr = pack_sockaddr_in $port, $ip_address

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

($port, $ip_address) = unpack_sockaddr_in $sockaddr

Takes a `sockaddr_in` structure (as returned by pack_sockaddr_in(), getpeername() or recv()). Returns a list 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 represent an `AF_INET` address.

In scalar context will return just the IP address.

$sockaddr = sockaddr_in $port, $ip_address

($port, $ip_address) = sockaddr_in $sockaddr

A wrapper of pack_sockaddr_in() or unpack_sockaddr_in(). In list context, unpacks its argument and returns a list consisting of the port and IP address. In scalar context, packs its port and IP address arguments as a `sockaddr_in` and returns it.

Provided largely for legacy compatibility; it is better to use pack_sockaddr_in() or unpack_sockaddr_in() explicitly.

$sockaddr = 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 and `AF_INET6` filled in. IPv6 equivalent of pack_sockaddr_in().

($port, $ip6_address, $scope_id, $flowinfo) = unpack_sockaddr_in6 $sockaddr

Takes a `sockaddr_in6` structure. Returns a list 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 represent an `AF_INET6` address.

In scalar context will return just the IP address.

$sockaddr = sockaddr_in6 $port, $ip6_address, [$scope_id, [$flowinfo]]

($port, $ip6_address, $scope_id, $flowinfo) = sockaddr_in6 $sockaddr

A wrapper of pack_sockaddr_in6() or unpack_sockaddr_in6(). In list context, unpacks its argument according to unpack_sockaddr_in6(). In scalar context, packs its arguments according to pack_sockaddr_in6().

Provided largely for legacy compatibility; it is better to use pack_sockaddr_in6() or unpack_sockaddr_in6() explicitly.

$sockaddr = 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 `PF_UNIX` sockets, this structure is normally what you need for the arguments in bind(), connect(), and send().

($path) = unpack_sockaddr_un $sockaddr

Takes a `sockaddr_un` structure (as returned by pack_sockaddr_un(), getpeername() or recv()). Returns a list of one element: the pathname. Will croak if the structure does not represent an `AF_UNIX` address.

$sockaddr = sockaddr_un $path

($path) = sockaddr_un $sockaddr

A wrapper of pack_sockaddr_un() or unpack_sockaddr_un(). In a list context, unpacks its argument and returns a list consisting of the pathname. In a scalar context, packs its pathname as a `sockaddr_un` and returns it.

Provided largely for legacy compatibility; it is better to use pack_sockaddr_un() or unpack_sockaddr_un() explicitly.

These are only supported if your system has <*sys/un.h*>.

$ip_mreq = pack_ip_mreq $multiaddr, $interface

Takes an IPv4 multicast address and optionally an interface address (or `INADDR_ANY`). Returns the `ip_mreq` structure with those arguments packed in. Suitable for use with the `IP_ADD_MEMBERSHIP` and `IP_DROP_MEMBERSHIP` sockopts.

($multiaddr, $interface) = unpack_ip_mreq $ip_mreq

Takes an `ip_mreq` structure. Returns a list of two elements; the IPv4 multicast address and interface address.

$ip_mreq_source = pack_ip_mreq_source $multiaddr, $source, $interface

Takes an IPv4 multicast address, source address, and optionally an interface address (or `INADDR_ANY`). Returns the `ip_mreq_source` structure with those arguments packed in. Suitable for use with the `IP_ADD_SOURCE_MEMBERSHIP` and `IP_DROP_SOURCE_MEMBERSHIP` sockopts.

($multiaddr, $source, $interface) = unpack_ip_mreq_source $ip_mreq

Takes an `ip_mreq_source` structure. Returns a list of three elements; the IPv4 multicast address, source address and interface address.

$ipv6_mreq = pack_ipv6_mreq $multiaddr6, $ifindex

Takes an IPv6 multicast address and an interface number. Returns the `ipv6_mreq` structure with those arguments packed in. Suitable for use with the `IPV6_ADD_MEMBERSHIP` and `IPV6_DROP_MEMBERSHIP` sockopts.

($multiaddr6, $ifindex) = unpack_ipv6_mreq $ipv6_mreq

Takes an `ipv6_mreq` structure. Returns a list of two elements; the IPv6 address and an interface number.

FUNCTIONS

$ip_address = inet_aton $string

Takes a string giving the name of a host, or a textual representation of an IP address and translates that to an packed binary address structure suitable to pass to pack_sockaddr_in(). If passed a hostname that 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.

This IPv4-only function is provided largely for legacy reasons. Newly-written code should use getaddrinfo() or inet_pton() instead for IPv6 support.

$string = inet_ntoa $ip_address

Takes a packed binary address structure such as returned by unpack_sockaddr_in() (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 ds are numbers less than 256 (the normal human-readable four dotted number notation for Internet addresses).

This IPv4-only function is provided largely for legacy reasons. Newly-written code should use getnameinfo() or inet_ntop() instead for IPv6 support.

$address = inet_pton $family, $string

Takes an address family (such as `AF_INET` or `AF_INET6`) and a string containing a textual representation of an address in that family and translates that to an packed binary address structure.

See also getaddrinfo() for a more powerful and flexible function to look up socket addresses given hostnames or textual addresses.

$string = inet_ntop $family, $address

Takes an address family and a packed binary address structure and translates it into a human-readable textual representation of the address; typically in `d.d.d.d` form for `AF_INET` or `hhhh:hhhh::hhhh` form for `AF_INET6`.

See also getnameinfo() for a more powerful and flexible function to turn socket addresses into human-readable textual representations.

($err, @result) = getaddrinfo $host, $service, [$hints]

Given both a hostname and service name, this function attempts to resolve the host name into a list of network addresses, and the service name into a protocol and port number, and then returns a list of address structures suitable to connect() to it.

Given just a host name, this function attempts to resolve it to a list of network addresses, and then returns a list of address structures giving these addresses.

Given just a service name, this function attempts to resolve it to a protocol and port number, and then returns a list of address structures that represent it suitable to bind() to. This use should be combined with the `AI_PASSIVE` flag; see below.

Given neither name, it generates an error.

If present, $hints should be a reference to a hash, where the following keys are recognised:

flags => INT

A bitfield containing `AI_*` constants; see below.

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 occurred).

The error value will be a dualvar; comparable to the `EI_*` error constants, or printable as a human-readable error message string. If no error occurred it will be zero numerically and an empty 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.

The following flag constants are recognised in the $hints hash. Other flag constants may exist as provided by the OS.

AI_PASSIVE

  Indicates that this resolution is for a local bind() for a passive (i.e. listening) socket, rather than an active (i.e. connecting) socket.

AI_CANONNAME

  Indicates that the caller wishes the canonical hostname (`canonname`) field of the result to be filled in.

AI_NUMERICHOST

  Indicates that the caller will pass a numeric address, rather than a hostname, and that getaddrinfo() must not perform a resolve operation on this name. This flag will prevent a possibly-slow network lookup operation, and instead return an error if a hostname is passed.

($err, $hostname, $servicename) = getnameinfo $sockaddr, [$flags, [$xflags]]

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.

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.

The following flag constants are recognised as $flags. Other flag constants may exist as provided by the OS.

NI_NUMERICHOST

  Requests that a human-readable string representation of the numeric address be returned directly, rather than performing a name resolve operation that may convert it into a hostname. This will also avoid potentially-blocking network IO.

NI_NUMERICSERV

  Requests that the port number be returned directly as a number representation rather than performing a name resolve operation that may convert it into a service name.

NI_NAMEREQD

If a name resolve operation fails to provide a name, then this flag will cause getnameinfo() to indicate an error, rather than returning the numeric representation as a human-readable string.

NI_DGRAM

Indicates that the socket address relates to a `SOCK_DGRAM` socket, for the services whose name differs between TCP and UDP protocols.

The following constants may be supplied as $xflags.

NIx_NOHOST

Indicates that the caller is not interested in the hostname of the result, so it does not have to be converted. `undef` will be returned as the hostname.

NIx_NOSERV

Indicates that the caller is not interested in the service name of the result, so it does not have to be converted. `undef` will be returned as the service name.

getaddrinfo() / getnameinfo() ERROR CONSTANTS

The following constants may be returned by getaddrinfo() or getnameinfo(). Others may be provided by the OS.

EAI_AGAIN

A temporary failure occurred during name resolution. The operation may be successful if it is retried later.

EAI_BADFLAGS

The value of the `flags` hint to getaddrinfo(), or the $flags parameter to getnameinfo() contains unrecognised flags.

EAI_FAMILY

The `family` hint to getaddrinfo(), or the family of the socket address passed to getnameinfo() is not supported.

EAI_NODATA

The host name supplied to getaddrinfo() did not provide any usable address data.

EAI_NONAME

The host name supplied to getaddrinfo() does not exist, or the address supplied to getnameinfo() is not associated with a host name and the `NI_NAMEREQD` flag was supplied.

EAI_SERVICE

The service name supplied to getaddrinfo() is not available for the socket type given in the $hints.

EXAMPLES

Lookup for connect()

The getaddrinfo() function converts a hostname and a service name into a list of structures, each containing a potential way to connect() to the named service on the named host.

```
use IO::Socket;
use Socket qw(SOCK_STREAM getaddrinfo);

my %hints = (socktype => SOCK_STREAM);
my ($err, @res) = getaddrinfo("localhost", "echo", \%hints);
die "Cannot getaddrinfo - $err" if $err;
```

```
 my $sock;

 foreach my $ai (@res) {
     my $candidate = IO::Socket->new();

     $candidate->socket($ai->{family}, $ai->{socktype}, $ai->{protocol})
         or next;

     $candidate->connect($ai->{addr})
         or next;

     $sock = $candidate;
     last;
 }

 die "Cannot connect to localhost:echo" unless $sock;

 $sock->print("Hello, world!\n");
 print <$sock>;
```

Because a list of potential candidates is returned, the `while` loop tries each in turn until it finds one that succeeds both the socket() and connect() calls.

This function performs the work of the legacy functions gethostbyname(), getservbyname(), inet_aton() and pack_sockaddr_in().

In practice this logic is better performed by *IO::Socket::IP*.

Making a human-readable string out of an address

The getnameinfo() function converts a socket address, such as returned by getsockname() or getpeername(), into a pair of human-readable strings representing the address and service name.

```
 use IO::Socket::IP;
 use Socket qw(getnameinfo);

 my $server = IO::Socket::IP->new(LocalPort => 12345, Listen => 1) or
     die "Cannot listen - $@";

 my $socket = $server->accept or die "accept: $!";

 my ($err, $hostname, $servicename) = getnameinfo($socket->peername);
 die "Cannot getnameinfo - $err" if $err;

 print "The peer is connected from $hostname\n";
```

Since in this example only the hostname was used, the redundant conversion of the port number into a service name may be omitted by passing the `NIx_NOSERV` flag.

```
 use Socket qw(getnameinfo NIx_NOSERV);

 my ($err, $hostname) = getnameinfo($socket->peername, 0, NIx_NOSERV);
```

This function performs the work of the legacy functions unpack_sockaddr_in(), inet_ntoa(), gethostbyaddr() and getservbyport().

In practice this logic is better performed by *IO::Socket::IP*.

Resolving hostnames into IP addresses

To turn a hostname into a human-readable plain IP address use getaddrinfo() to turn the hostname into a list of socket structures, then getnameinfo() on each one to make it a readable IP address again.

```
 use Socket qw(:addrinfo SOCK_RAW);

 my ($err, @res) = getaddrinfo($hostname, "", {socktype => SOCK_RAW});
 die "Cannot getaddrinfo - $err" if $err;

 while( my $ai = shift @res ) {
     my ($err, $ipaddr) = getnameinfo($ai->{addr}, NI_NUMERICHOST,
NIx_NOSERV);
     die "Cannot getnameinfo - $err" if $err;

     print "$ipaddr\n";
 }
```

The `socktype` hint to getaddrinfo() filters the results to only include one socket type and protocol. Without this most OSes return three combinations, for `SOCK_STREAM`, `SOCK_DGRAM` and `SOCK_RAW`, resulting in triplicate output of addresses. The `NI_NUMERICHOST` flag to getnameinfo() causes it to return a string-formatted plain IP address, rather than reverse resolving it back into a hostname.

This combination performs the work of the legacy functions gethostbyname() and inet_ntoa().

Accessing socket options

The many `SO_*` and other constants provide the socket option names for getsockopt() and setsockopt().

```
 use IO::Socket::INET;
 use Socket qw(SOL_SOCKET SO_RCVBUF IPPROTO_IP IP_TTL);

 my $socket = IO::Socket::INET->new(LocalPort => 0, Proto => 'udp')
     or die "Cannot create socket: $@";

 $socket->setsockopt(SOL_SOCKET, SO_RCVBUF, 64*1024) or
     die "setsockopt: $!";

 print "Receive buffer is ", $socket->getsockopt(SOL_SOCKET, SO_RCVBUF),
     " bytes\n";

 print "IP TTL is ", $socket->getsockopt(IPPROTO_IP, IP_TTL), "\n";
```

As a convenience, *IO::Socket*'s setsockopt() method will convert a number into a packed byte buffer, and getsockopt() will unpack a byte buffer of the correct size back into a number.

AUTHOR

This module was originally maintained in Perl core by the Perl 5 Porters.

It was extracted to dual-life on CPAN at version 1.95 by Paul Evans <leonerd@leonerd.org.uk>