

NAME

IO::Socket - Object interface to socket communications

SYNOPSIS

```
use IO::Socket;
```

DESCRIPTION

`IO::Socket` provides an object interface to creating and using sockets. It is built upon the `IO::Handle` interface and inherits all the methods defined by `IO::Handle`.

`IO::Socket` only defines methods for those operations which are common to all types of socket. Operations which are specified to a socket in a particular domain have methods defined in sub classes of `IO::Socket`

`IO::Socket` will export all functions (and constants) defined by `Socket`.

CONSTRUCTOR

```
new ( [ARGS] )
```

Creates an `IO::Socket`, which is a reference to a newly created symbol (see the `Symbol` package). `new` optionally takes arguments, these arguments are in key-value pairs. `new` only looks for one key `Domain` which tells `new` which domain the socket will be in. All other arguments will be passed to the configuration method of the package for that domain, See below.

NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE

As of VERSION 1.18 all `IO::Socket` objects have autoflush turned on by default. This was not the case with earlier releases.

NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE

METHODS

See *perlfunc* for complete descriptions of each of the following supported `IO::Socket` methods, which are just front ends for the corresponding built-in functions:

```
socket
socketpair
bind
listen
accept
send
recv
peername (getpeername)
sockname (getsockname)
shutdown
```

Some methods take slightly different arguments to those defined in *perlfunc* in attempt to make the interface more flexible. These are

```
accept([PKG])
```

perform the system call `accept` on the socket and return a new object. The new object will be created in the same class as the listen socket, unless `PKG` is specified. This object can be used to communicate with the client that was trying to connect.

In a scalar context the new socket is returned, or `undef` upon failure. In a list context a two-element array is returned containing the new socket and the peer address; the list will be empty upon failure.

The timeout in the [PKG] can be specified as zero to effect a "poll", but you shouldn't do that because a new IO::Select object will be created behind the scenes just to do the single poll. This is horrendously inefficient. Use rather `true select()` with a zero timeout on the handle, or non-blocking IO.

`socketpair(DOMAIN, TYPE, PROTOCOL)`

Call `socketpair` and return a list of two sockets created, or an empty list on failure.

Additional methods that are provided are:

`atmark`

True if the socket is currently positioned at the urgent data mark, false otherwise.

```
use IO::Socket;
```

```
my $sock = IO::Socket::INET->new('some_server');
$sock->read($data, 1024) until $sock->atmark;
```

Note: this is a reasonably new addition to the family of socket functions, so all systems may not support this yet. If it is unsupported by the system, an attempt to use this method will abort the program.

The `atmark()` functionality is also exportable as `socketatmark()` function:

```
use IO::Socket 'socketatmark';
```

This allows for a more traditional use of `socketatmark()` as a procedural socket function. If your system does not support `socketatmark()`, the `use` declaration will fail at compile time.

`connected`

If the socket is in a connected state, the peer address is returned. If the socket is not in a connected state, `undef` is returned.

Note that `connected()` considers a half-open TCP socket to be "in a connected state". Specifically, `connected()` does not distinguish between the **ESTABLISHED** and **CLOSE-WAIT** TCP states; it returns the peer address, rather than `undef`, in either case. Thus, in general, `connected()` cannot be used to reliably learn whether the peer has initiated a graceful shutdown because in most cases (see below) the local TCP state machine remains in **CLOSE-WAIT** until the local application calls `shutdown()` or `close()`; only at that point does `connected()` return `undef`.

The "in most cases" hedge is because local TCP state machine behavior may depend on the peer's socket options. In particular, if the peer socket has `SO_LINGER` enabled with a zero timeout, then the peer's `close()` will generate a RST segment, upon receipt of which the local TCP transitions immediately to **CLOSED**, and in that state, `connected()` *will* return `undef`.

`protocol`

Returns the numerical number for the protocol being used on the socket, if known. If the protocol is unknown, as with an `AF_UNIX` socket, zero is returned.

`socketdomain`

Returns the numerical number for the socket domain type. For example, for an `AF_INET` socket the value of `&AF_INET` will be returned.

`sockopt(OPT [, VAL])`

Unified method to both set and get options in the `SOL_SOCKET` level. If called with one argument then `getsockopt` is called, otherwise `setsockopt` is called.

`getsockopt(LEVEL, OPT)`

Get option associated with the socket. Other levels than `SOL_SOCKET` may be specified

`setsockopt(opt(LEVEL, OPT, VAL)`

Set option associated with the socket. Other levels than SOL_SOCKET may be specified here.

`sockettype`

Returns the numerical number for the socket type. For example, for a SOCK_STREAM socket the value of &SOCK_STREAM will be returned.

`timeout([VAL])`

Set or get the timeout value (in seconds) associated with this socket. If called without any arguments then the current setting is returned. If called with an argument the current setting is changed and the previous value returned.

LIMITATIONS

On some systems, for an IO::Socket object created with `new_from_fd()`, or created with `accept()` from such an object, the `protocol()`, `sockdomain()` and `sockettype()` methods may return undef.

SEE ALSO

Socket, *IO::Handle*, *IO::Socket::INET*, *IO::Socket::UNIX*

AUTHOR

Graham Barr. `atmark()` by Lincoln Stein. Currently maintained by the Perl Porters. Please report all bugs to <perlbug@perl.org>.

COPYRIGHT

Copyright (c) 1997-8 Graham Barr <gbarr@pobox.com>. All rights reserved. This program is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

The `atmark()` implementation: Copyright 2001, Lincoln Stein <lstein@cshl.org>. This module is distributed under the same terms as Perl itself. Feel free to use, modify and redistribute it as long as you retain the correct attribution.