# Sockets

`Sockets.Sockets` — Module

Support for sockets. Provides `IPAddr` and subtypes, `TCPSocket`, and `UDPSocket`.

`Sockets.connect` — Method

```
connect([host], port::Integer) -> TCPSocket
```

Connect to the host `host` on port `port`.

`Sockets.connect` — Method

```
connect(path::AbstractString) -> PipeEndpoint
```

Connect to the named pipe / UNIX domain socket at `path`.

`Sockets.listen` — Method

```
listen([addr, ]port::Integer; backlog::Integer=BACKLOG_DEFAULT) -> TCPServer
```

Listen on port on the address specified by `addr`. By default this listens on `localhost` only. To listen on all interfaces pass `IPv4(0)` or `IPv6(0)` as appropriate. `backlog` determines how many connections can be pending (not having called `accept`) before the server will begin to reject them. The default value of `backlog` is 511.

`Sockets.listen` — Method

```
listen(path::AbstractString) -> PipeServer
```

Create and listen on a named pipe / UNIX domain socket.

### Sockets.getaddrinfo — Function

```
getaddrinfo(host::AbstractString, IPAddr=IPv4) -> IPAddr
```

Gets the first IP address of the `host` of the specified `IPAddr` type. Uses the operating system's underlying getaddrinfo implementation, which may do a DNS lookup.

### Sockets.getipaddr — Function

```
getipaddr() -> IPAddr
```

Get an IP address of the local machine, preferring IPv4 over IPv6. Throws if no addresses are available.

```
getipaddr(addr_type::Type{T}) where T<:IPAddr -> T
```

Get an IP address of the local machine of the specified type. Throws if no addresses of the specified type are available.

This function is a backwards-compatibility wrapper around `getipaddrs`. New applications should use `getipaddrs` instead.

Examples

```
julia> getipaddr()
ip"192.168.1.28"

julia> getipaddr(IPv6)
ip"fe80::9731:35af:e1c5:6e49"
```

See also: `getipaddrs`

### Sockets.getipaddrs — Function

```
getipaddrs(addr_type::Type{T}=IPAddr; loopback::Bool=false) where T<:IPAddr ->
```

Get the IP addresses of the local machine.

Setting the optional `addr_type` parameter to `IPv4` or `IPv6` causes only addresses of that type to be returned.

The `loopback` keyword argument dictates whether loopback addresses (e.g. `ip"127.0.0.1"`, `ip"::1"`) are included.

> **Julia 1.2**
>
> This function is available as of Julia 1.2.

Examples

```
julia> getipaddrs()
5-element Array{IPAddr,1}:
 ip"198.51.100.17"
 ip"203.0.113.2"
 ip"2001:db8:8:4:445e:5fff:fe5d:5500"
 ip"2001:db8:8:4:c164:402e:7e3c:3668"
 ip"fe80::445e:5fff:fe5d:5500"

julia> getipaddrs(IPv6)
3-element Array{IPv6,1}:
 ip"2001:db8:8:4:445e:5fff:fe5d:5500"
 ip"2001:db8:8:4:c164:402e:7e3c:3668"
 ip"fe80::445e:5fff:fe5d:5500"
```

See also: `islinklocaladdr`, `split(ENV["SSH_CONNECTION"], ' ')[3]`

### Sockets.islinklocaladdr — Function

```
islinklocaladdr(addr::IPAddr)
```

Tests if an IP address is a link-local address. Link-local addresses are not guaranteed to be unique beyond their network segment, therefore routers do not forward them. Link-local addresses are from the address blocks `169.254.0.0/16` or `fe80::/10`.

Example

```
filter(!islinklocaladdr, getipaddrs())
```

### `Sockets.getalladdrinfo` — Function

```
getalladdrinfo(host::AbstractString) -> Vector{IPAddr}
```

Gets all of the IP addresses of the `host`. Uses the operating system's underlying `getaddrinfo` implementation, which may do a DNS lookup.

Example

```
julia> getalladdrinfo("google.com")
2-element Array{IPAddr,1}:
 ip"172.217.6.174"
 ip"2607:f8b0:4000:804::200e"
```

### `Sockets.DNSError` — Type

```
DNSError
```

The type of exception thrown when an error occurs in DNS lookup. The `host` field indicates the host URL string. The `code` field indicates the error code based on libuv.

### `Sockets.getnameinfo` — Function

```
getnameinfo(host::IPAddr) -> String
```

Performs a reverse-lookup for IP address to return a hostname and service using the operating system's underlying `getnameinfo` implementation.

Examples

```
julia> getnameinfo(Sockets.IPv4("8.8.8.8"))
"google-public-dns-a.google.com"
```

### Sockets.getsockname — Function

```
getsockname(sock::Union{TCPServer, TCPSocket}) -> (IPAddr, UInt16)
```

Get the IP address and port that the given socket is bound to.

### Sockets.getpeername — Function

```
getpeername(sock::TCPSocket) -> (IPAddr, UInt16)
```

Get the IP address and port of the remote endpoint that the given socket is connected to. Valid only for connected TCP sockets.

### Sockets.IPAddr — Type

```
IPAddr
```

Abstract supertype for IP addresses. `IPv4` and `IPv6` are subtypes of this.

### Sockets.IPv4 — Type

```
IPv4(host::Integer) -> IPv4
```

Returns an IPv4 object from ip address `host` formatted as an `Integer`.

Examples

```
julia> IPv4(3223256218)
```

```
ip"192.30.252.154"
```

### Sockets.IPv6 — Type

```
IPv6(host::Integer) -> IPv6
```

Returns an IPv6 object from ip address `host` formatted as an `Integer`.

Examples

```
julia> IPv6(3223256218)
ip"::c01e:fc9a"
```

### Sockets.@ip_str — Macro

```
@ip_str str -> IPAddr
```

Parse `str` as an IP address.

Examples

```
julia> ip"127.0.0.1"
ip"127.0.0.1"

julia> @ip_str "2001:db8:0:0:0:0:2:1"
ip"2001:db8::2:1"
```

### Sockets.TCPSocket — Type

```
TCPSocket(; delay=true)
```

Open a TCP socket using libuv. If `delay` is true, libuv delays creation of the socket's file descriptor till the first `bind` call. `TCPSocket` has various fields to denote the state of the socket as well as its send/receive buffers.

### Sockets.UDPSocket — Type

```
UDPSocket()
```

Open a UDP socket using libuv. `UDPSocket` has various fields to denote the state of the socket.

### Sockets.accept — Function

```
accept(server[, client])
```

Accepts a connection on the given server and returns a connection to the client. An uninitialized client stream may be provided, in which case it will be used instead of creating a new stream.

### Sockets.listenany — Function

```
listenany([host::IPAddr,] port_hint) -> (UInt16, TCPServer)
```

Create a `TCPServer` on any port, using hint as a starting point. Returns a tuple of the actual port that the server was created on and the server itself.

### Base.bind — Function

```
bind(socket::Union{UDPSocket, TCPSocket}, host::IPAddr, port::Integer; ipv6only
```

Bind `socket` to the given `host:port`. Note that `0.0.0.0` will listen on all devices.

- The `ipv6only` parameter disables dual stack mode. If `ipv6only=true`, only an IPv6 stack is created.
- If `reuseaddr=true`, multiple threads or processes can bind to the same address without error if they all set `reuseaddr=true`, but only the last to bind will receive any traffic.

```
bind(chnl::Channel, task::Task)
```

Associate the lifetime of `chnl` with a task. `Channel chnl` is automatically closed when the task terminates. Any uncaught exception in the task is propagated to all waiters on `chnl`.

The `chnl` object can be explicitly closed independent of task termination. Terminating tasks have no effect on already closed `Channel` objects.

When a channel is bound to multiple tasks, the first task to terminate will close the channel. When multiple channels are bound to the same task, termination of the task will close all of the bound channels.

Examples

```
julia> c = Channel(0);

julia> task = @async foreach(i->put!(c, i), 1:4);

julia> bind(c,task);

julia> for i in c
           @show i
       end;
i = 1
i = 2
i = 3
i = 4

julia> isopen(c)
false
```

```
julia> c = Channel(0);

julia> task = @async (put!(c, 1); error("foo"));

julia> bind(c, task);

julia> take!(c)
1

julia> put!(c, 1);
ERROR: TaskFailedException:
foo
Stacktrace:
[...]
```

### Sockets.send — Function

```
send(socket::UDPSocket, host::IPAddr, port::Integer, msg)
```

Send `msg` over `socket` to `host:port`.

### Sockets.recv — Function

```
recv(socket::UDPSocket)
```

Read a UDP packet from the specified socket, and return the bytes received. This call blocks.

### Sockets.recvfrom — Function

```
recvfrom(socket::UDPSocket) -> (host_port, data)
```

Read a UDP packet from the specified socket, returning a tuple of `(host_port, data)`, where `host_port` will be an InetAddr{IPv4} or InetAddr{IPv6}, as appropriate.

> **Julia 1.3**
>
> Prior to Julia version 1.3, the first returned value was an address (`IPAddr`). In version 1.3 it was changed to an `InetAddr`.

### Sockets.setopt — Function

```
setopt(sock::UDPSocket; multicast_loop=nothing, multicast_ttl=nothing, enable_b
```

Set UDP socket options.

- `multicast_loop`: loopback for multicast packets (default: `true`).
- `multicast_ttl`: TTL for multicast packets (default: `nothing`).
- `enable_broadcast`: flag must be set to `true` if socket will be used for broadcast messages, or

else the UDP system will return an access error (default: `false`).

- `ttl`: Time-to-live of packets sent on the socket (default: `nothing`).

### `Sockets.nagle` — Function

```
nagle(socket::Union{TCPServer, TCPSocket}, enable::Bool)
```

Enables or disables Nagle's algorithm on a given TCP server or socket.

### `Sockets.quickack` — Function

```
quickack(socket::Union{TCPServer, TCPSocket}, enable::Bool)
```

On Linux systems, the TCP_QUICKACK is disabled or enabled on `socket`.

« Shared Arrays Sparse Arrays »