IOLib 0.6.0 Manual

draft version

Table of Contents

1 Overview

Describe IOLib here.

2 Networking

2.1 Overview

Describe networking here.

2.2 Sockets

2.2.1 Socket Classes

`sockets:socket` [Class]

Class precedence list: `socket`, `dual-channel-single-fd-mixin`, `dual-channel-fd-mixin`, `standard-object`, `t`

Base class for sockets.

`sockets:stream-socket` [Class]

Class precedence list: `stream-socket`, `socket`, `dual-channel-single-fd-mixin`, `dual-channel-fd-mixin`, `standard-object`, `t`

Mixin for sockets of type SOCK_STREAM.

`sockets:datagram-socket` [Class]

Class precedence list: `datagram-socket`, `socket`, `dual-channel-single-fd-mixin`, `dual-channel-fd-mixin`, `standard-object`, `t`

Mixin for sockets of type SOCK_DGRAM.

`sockets:internet-socket` [Class]

Class precedence list: `internet-socket`, `socket`, `dual-channel-single-fd-mixin`, `dual-channel-fd-mixin`, `standard-object`, `t`

Mixin for sockets of domain AF_INET or AF_INET6.

`sockets:local-socket` [Class]

Class precedence list: `local-socket`, `socket`, `dual-channel-single-fd-mixin`, `dual-channel-fd-mixin`, `standard-object`, `t`

Mixin for sockets of domain AF_LOCAL.

`sockets:active-socket` [Class]

Class precedence list: `active-socket`, `socket`, `dual-channel-single-fd-mixin`, `dual-channel-gray-stream`, `trivial-gray-stream-mixin`, `dual-channel-fd-mixin`, `fundamental-binary-input-stream`, `fundamental-binary-output-stream`, `fundamental-character-input-stream`, `fundamental-input-stream`, `fundamental-character-output-stream`, `fundamental-output-stream`, `fundamental-character-stream`, `fundamental-binary-stream`, `fundamental-stream`, `standard-object`, `stream`, `t`

Mixin class for active(client) sockets.

`sockets:passive-socket` [Class]

Class precedence list: `passive-socket`, `socket`, `dual-channel-single-fd-mixin`, `dual-channel-fd-mixin`, `standard-object`, `t`

Mixin class for PASSIVE(server) sockets.

`sockets:socket-stream-internet-active` [Class]

Class precedence list: `socket-stream-internet-active, active-socket, stream-socket, internet-socket, socket, dual-channel-single-fd-mixin, dual-channel-gray-stream, trivial-gray-stream-mixin, dual-channel-fd-mixin, fundamental-binary-input-stream, fundamental-binary-output-stream, fundamental-character-input-stream, fundamental-input-stream, fundamental-character-output-stream, fundamental-output-stream, fundamental-character-stream, fundamental-binary-stream, fundamental-stream, standard-object, stream, t`

Class representing active sockets of type SOCK_STREAM and domain AF_INET or AF_INET6.

`sockets:socket-stream-internet-passive` [Class]

Class precedence list: `socket-stream-internet-passive, passive-socket, stream-socket, internet-socket, socket, dual-channel-single-fd-mixin, dual-channel-fd-mixin, standard-object, t`

Class representing passive sockets of type SOCK_STREAM and domain AF_INET or AF_INET6.

`sockets:socket-stream-local-active` [Class]

Class precedence list: `socket-stream-local-active, active-socket, stream-socket, local-socket, socket, dual-channel-single-fd-mixin, dual-channel-gray-stream, trivial-gray-stream-mixin, dual-channel-fd-mixin, fundamental-binary-input-stream, fundamental-binary-output-stream, fundamental-character-input-stream, fundamental-input-stream, fundamental-character-output-stream, fundamental-output-stream, fundamental-character-stream, fundamental-binary-stream, fundamental-stream, standard-object, stream, t`

Class representing active sockets of type SOCK_STREAM and domain AF_LOCAL.

`sockets:socket-stream-local-passive` [Class]

Class precedence list: `socket-stream-local-passive, passive-socket, stream-socket, local-socket, socket, dual-channel-single-fd-mixin, dual-channel-fd-mixin, standard-object, t`

Class representing passive sockets of type SOCK_STREAM and domain AF_LOCAL.

`sockets:socket-datagram-internet-active` [Class]

Class precedence list: `socket-datagram-internet-active, active-socket, datagram-socket, internet-socket, socket, dual-channel-single-fd-mixin, dual-channel-gray-stream, trivial-gray-stream-mixin, dual-channel-fd-mixin, fundamental-binary-input-stream, fundamental-binary-output-stream, fundamental-character-input-stream, fundamental-input-stream, fundamental-character-output-stream, fundamental-output-stream, fundamental-character-stream, fundamental-binary-stream, fundamental-stream, standard-object, stream, t`

Class representing active sockets of type SOCK_DGRAM and domain AF_INET or AF_INET6.

`sockets:socket-datagram-local-active` [Class]

Class precedence list: `socket-datagram-local-active`, `active-socket`, `datagram-socket`, `local-socket`, `socket`, `dual-channel-single-fd-mixin`, `dual-channel-gray-stream`, `trivial-gray-stream-mixin`, `dual-channel-fd-mixin`, `fundamental-binary-input-stream`, `fundamental-binary-output-stream`, `fundamental-character-input-stream`, `fundamental-input-stream`, `fundamental-character-output-stream`, `fundamental-output-stream`, `fundamental-character-stream`, `fundamental-binary-stream`, `fundamental-stream`, `standard-object`, `stream`, `t`

Class representing active sockets of type SOCK_DGRAM and domain AF_LOCAL.

2.2.2 Socket Operators and Macros

`sockets:make-socket` **&key** *address-family type connect ipv6* [Generic Function]
external-format address-family type connect ipv6 external-format
&allow-other-keys

Create an instance of a subclass of `socket`. `address-family`, `type`, `connect` and IPV6 are used to specify the kind of socket to create.

- `address-family` - `:internet` or `:local` (or `:file` as synonim)
- `type` - `:stream` or `:datagram`
- `connect` - `:active` or `:passive`
- IPV6 - if `nil`, create an IPv4 socket, otherwise an IPv6 socket.

To initialize the socket, the following keyword arguments can be used depending on `address-family`, `type` and `connect`:

- :local-host - a hostname designator or `nil`. If non-null the socket will be bound to this address
- :local-port - a port designator or `nil`. If `local-host` is non-null, bind the socket to this port. If `nil`, choose a random port
- :remote-host - a hostname designator or `nil`. If non-null the socket will be connected to this address
- :remote-port - a port designator. If `remote-host` is non-null, connect the socket to this port
- :local-filename - a string or `nil`. If non-null the socket will be bound to this file
- :remote-filename - a string or `nil`. If non-null the socket will be connected to this file
- :backlog - a positive integer or `nil`. Specifies the length of the incomming connection queue and can't be larger than `+max-backlog-size+`. If `nil`, default is `*default-backlog-size*`
- :reuse-address: a boolean(default T). set option SO_REUSEADDR if `local-host` is non-null
- :keepalive - a boolean. set option SO_KEEPALIVE
- :nodelay - a boolean. set option SO_NODELAY
- :interface - a string. set option SO_BINDTODEVICE to this interface

- :input-buffer-size - a positive integer. Create the stream input buffer of this size
- :output-buffer-size - a positive integer. Create the stream output buffer of this size

Glossary:

- hostname designator: an instance of `inet-address` or any object accepted by `lookup-host`. IPV6 is passed to `lookup-host` as is
- port designator: any object accepted by `lookup-service`

:address-family `:internet` :type `:stream` :connect `:active`

- Valid keyword args: `:local-host`, `:local-port`, `:remote-host`, `:remote-port`, `:reuse-address`, `:keepalive`, `:nodelay`, `:input-buffer-size` and `:output-buffer-size`

:address-family `:internet` :type `:stream` :connect `:passive`

- Valid keyword args: `:local-host`, `:local-port`, `:backlog`, `:reuse-address`, `:interface` and `:nodelay`

:address-family `:internet` :type `:stream` :connect `:active`

- Valid keyword args: `:local-filename`, `:remote-filename`, `:input-buffer-size` and `:output-buffer-size`

:address-family `:internet` :type `:stream` :connect `:passive`

- Valid keyword args: `:local-filename`, `:remote-filename`, `:backlog` and `:reuse-address`

:address-family `:internet` :type `:datagram`

- Valid keyword args: `:local-host`, `:local-port`, `:remote-host`, `:remote-port`, `:reuse-address`, `:interface` and `:broadcast`

:address-family `:local` :type `:datagram`

- Valid keyword args: `:local-filename` and `:remote-filename`

`sockets:with-open-socket` (*var* **&rest** *args*) **&body** *body* [Macro]

Bind `var` to a socket created by passing `args` to `make-socket` and execute `body` as implicit `progn`. The socket is automatically closed upon exit.

`sockets:make-socket-from-fd` *fd* **&key** *connect* [Generic Function]
external-format input-buffer-size output-buffer-size connect external-format

Create a socket instance of the appropriate subclass of `socket` using `fd`. The connection type of the socket must be specified - `:active` or `:passive`. The address family and type of the socket are automatically discovered using `os` functions. Buffer sizes for the new socket can also be specified using `input-buffer-size` and `output-buffer-size`.

`sockets:make-socket-pair` **&key** *type protocol* [Generic Function]
external-format input-buffer-size output-buffer-size type protocol external-format

Create a pair of sockets connected to each other. The socket type must be either `:stream` or `:datagram`. Currently OSes can only create `:local` sockets this way. Buffer sizes for the new sockets can also be specified using `input-buffer-size` and `output-buffer-size`.

`sockets:send-to` *socket buffer* **&rest** *args* **&key** *start end remote-filename flags remote-host remote-port ipv6* **&allow-other-keys** [Generic Function]

Send the contents of `buffer` to `socket`. `buffer` must be a vector that can be coerced to a (SIMPLE-ARRAY (UNSIGNED-BYTE 8) (*)). `start` and `end` are used a bounding index on `buffer`. For disconnected datagram sockets, `remote-host` and `remote-port` or `remote-filename` are used as destination for the data.

Some flags can also be passed to sendto(2):

- `:out-of-band` for receiving out-of-band data - only for stream sockets
- `:dont-wait` for making only the current call non-blocking
- `:dont-route` for sending only to hosts on directly connected networks, not using gateways
- `:confirm` for signalling progress on the link layer - only available on Linux and only with `datagram` sockets
- `:more` for telling the kernel that there is more data to send - only available on Linux

Returns the number of bytes sent.

`sockets:receive-from` *socket* **&rest** *args* **&key** *flags end start buffer size* **&allow-other-keys** [Generic Function]

Receives data from `socket`. If `buffer` is specified `start` and `end` are used as bounding index. In that case `buffer` must be an array and its `array-element-type` be either (UNSIGNED-BYTE 8) or `t`. If `buffer` is not specified an (UNSIGNED-BYTE 8) buffer of size `size` will be allocated.

Some flags can also be passed to recvfrom(2):

- `:out-of-band` for receiving out-of-band data - only for `stream` sockets
- `:peek` for keeping the returned data in the kernel buffers
- `:wait-all` for waiting until the entire buffer can be filled
- `:dont-wait` for making only the current call non-blocking

The first two values returned are the buffer and the number of elements that have been copied into the buffer. For `internet datagram` sockets, two additional values are returned: the host and port of the remote peer from which the data was received. For `local datagram` sockets, one additional values is returned: the filename of the remote peer from which the data was received.

`sockets:bind-address` *socket address* **&key** *reuse-address port* **&allow-other-keys** [Generic Function]

Sets the local address of `socket` to ADDRESS(and `port` for `internet` sockets). `reuse-address` sets the SO_REUSEADDR socket option on `socket`.

`sockets:listen-on` *socket* **&key** *backlog* **&allow-other-keys** [Generic Function]

Start allowing incoming connections on `socket`. `backlog` specifies the maximum length of the queue of pending connections.

`sockets:accept-connection` *passive-socket* **&key** *timeout wait output-buffer-size input-buffer-size external-format* **&allow-other-keys** [Generic Function]

Returns one connection from the queue of pending connections on `socket`. If `wait` is true, waits until a connection is received or `timeout` expires in which case returns `nil`. If `wait` is false and there are no pending connections return `nil`. `external-format` optionally specifies the external format of the new socket - the default being that of `socket`. Buffer sizes for the new socket can also be specified using `input-buffer-size` and `output-buffer-size`.

`sockets:with-accept-connection` (*var passive-socket* **&rest** *args*) **&body** *body* [Macro]

Bind `var` to a socket created by passing `passive-socket` and `args` to `accept-connection` and execute `body` as implicit `progn`. The socket is automatically closed upon exit.

`sockets:connect` *socket address* **&key** *port wait timeout* **&allow-other-keys** [Generic Function]

Connects `socket` to `address`. For `internet` sockets you can specify the port to connect to using keyword argument `port`. The default value of `port` is 0, which usually means letting the `os` choose a random port to connect to. For `internet` sockets, if `wait` is true and a connection cannot be established within `timeout` seconds signal `iomux:poll-timeout`, but it works only with non-blocking sockets.

`sockets:disconnect` *socket* [Generic Function]

Disassociates `socket` from any remote address. Works only on `datagram` sockets.

`sockets:shutdown` *socket* **&key** *read write* [Generic Function]

Shut down all or part of a connection. If `read` it non-NIL, further receptions are disallowed; if `write` is non-NIL, further transmissions are disallowed. `close` must still be called on `socket` in order to release `os` resources.

`sockets:send-file-descriptor` *socket file-descriptor* [Generic Function]

Send `file-descriptor` through `socket`. The receiving process must use `receive-file-descriptor` to receive the file descriptor in order for it to be valid in the receiving process.

`sockets:receive-file-descriptor` *socket* [Generic Function]

Receive a file descriptor as ancillary data through `socket`.

2.2.3 Socket Accessors

`sockets:socket-os-fd` *socket* [Generic Function]

Returns the `os` file descriptor of `socket`.

`sockets:socket-type` *socket* [Generic Function]

Returns the socket type of `socket` - `:stream` or `:datagram`.

`sockets:socket-protocol` *object* [Generic Function]

Return the protocol of a socket.

`sockets:socket-address-family` *object* [Generic Function]
Return the address family of a socket.

`sockets:local-name` *socket* [Generic Function]
For `internet` sockets, returns two values: the local host and the local port. For `local` sockets, returns the local filename.

`sockets:local-host` *socket* [Generic Function]
Returns the local host of `socket`. Works only on `internet` sockets.

`sockets:local-port` *socket* [Generic Function]
Returns the local port of `socket` - an (UNSIGNED-BYTE 16). Works only on `internet` sockets.

`sockets:local-filename` *socket* [Generic Function]
Returns the local filename of `socket`. Works only on `local` sockets.

`sockets:remote-name` *socket* [Generic Function]
For `internet` sockets, returns two values: the remote host and the remote port. For `remote` sockets, returns the remote filename.

`sockets:remote-host` *socket* [Generic Function]
Returns the remote host of `socket`. Works only on `internet` sockets.

`sockets:remote-port` *socket* [Generic Function]
Returns the remote port of `socket` - an (UNSIGNED-BYTE 16). Works only on `internet` sockets.

`sockets:remote-filename` *socket* [Generic Function]
Returns the remote filename of `socket`. Works only on `local` sockets.

2.2.4 Socket Predicates

`sockets:socket-open-p` *socket* [Generic Function]
Returns a boolean indicating whether or not the file descriptor of `socket` is open.

`sockets:socket-connected-p` *socket* [Generic Function]
Returns a boolean specifying whether or not `socket` is connected.

`sockets:socket-ipv6-p` *socket* [Function]
Return `t` if `socket` is an AF_INET6 socket.

2.2.5 Socket Options

2.3 Socket Addresses

2.3.1 Address Objects

2.3.1.1 Classes

`sockets:address` [Class]
Class precedence list: `address, standard-object, t`

Base class for all socket address classes.

`sockets:inet-address` [Class]
Class precedence list: `inet-address, address, standard-object, t`

Base class for IPv4 and IPv6 addresses.

`sockets:ipv4-address` [Class]
Class precedence list: `ipv4-address, inet-address, address, standard-object, t`

IPv4 address. Its low-level representation can be accessed as vector of type IPV4-ARRAY through the `address-name` reader.

`sockets:ipv6-address` [Class]
Class precedence list: `ipv6-address, inet-address, address, standard-object, t`

IPv6 address. Its low-level representation can be accessed as vector of type IPV6-ARRAY through the `address-name` reader.

`sockets:local-address` [Class]
Class precedence list: `local-address, address, standard-object, t`

`unix` socket address.

2.3.1.2 Functions

`sockets:make-address` *name* [Function]
Constructs an `address` object. `name` should be of type IPV4-ARRAY, IPV6-ARRAY or `string` in which case an instance of IPV4-ADDRESS, IPV6-ADDRESS or `local-address`, respectively, will be created. Otherwise, a `type-error` is signalled. See also `ensure-address`.

`sockets:copy-address` *address* [Generic Function]
Returns a copy of `address` which is ADDRESS= to the original.

`sockets:ensure-address` *address* **&key** *family abstract errorp* [Function]
If `family` is `:local`, a `local-address` is instantiated with `address` as its `name` slot. If `family` is `:internet`, an appropriate subtype of `inet-address` is instantiated after guessing the address type through `address-to-vector`. If the address is invalid and `errorp` is not `nil`, then a `cl:parse-error` is signalled, otherwise `nil` is returned.

When `address` is already an instance of the `address` class, a check is made to see if it matches the `family` argument and it is returned unmodified.

`sockets:address-to-string` *address* [Generic Function]
Returns a textual presentation of `address`.

`sockets:address-to-vector` *address* [Function]
Convert any representation of an internet address to a vector. Allowed inputs are: unsigned 32-bit integers, strings, vectors and `inet-address` objects. If the address is valid, two values are returned: the vector and the address type (:IPV4 or IPV6), otherwise `nil` is returned.

2.3.1.3 Constants

`sockets:+ipv4-loopback+` [Constant]
Loopback IPv4 address. (127.0.0.1)

`sockets:+ipv4-unspecified+` [Constant]
Unspecified IPv4 address. (0.0.0.0)

`sockets:+ipv6-interface-local-all-nodes+` [Constant]
Interface local all nodes address. (ff01::1)

`sockets:+ipv6-interface-local-all-routers+` [Constant]
Interface local all routers address. (ff01::2)

`sockets:+ipv6-link-local-all-nodes+` [Constant]
Link local all nodes address. (ff02::1)

`sockets:+ipv6-link-local-all-routers+` [Constant]
Link local all routers address. (ff02::2)

`sockets:+ipv6-loopback+` [Constant]
Loopback IPv6 address. (::1)

`sockets:+ipv6-site-local-all-routers+` [Constant]
Site local all routers address. (ff05::2)

`sockets:+ipv6-unspecified+` [Constant]
Unspecified IPv6 address. (::)

2.3.2 Address Predicates

`sockets:addressp` *address* [Function]
Returns `t` if `address` is an object of class `address`. Does not return `t` for other low-level address representations.

`sockets:address=` *addr1 addr2* [Generic Function]
Returns `t` if both arguments are the same socket address.

`sockets:address-equal-p` *addr1 addr2* **&optional** *family* [Function]
Returns `t` if both arguments are designators for the same socket address.

`sockets:ipv4-address-p` *address* [Function]
Returns `t` if `address` is an IPv4 address object.

`sockets:ipv6-address-p` *address* [Function]
Returns `t` if `address` is an IPv6 address object.

`sockets:local-address-p` *address* [Function]
Returns `t` if `address` is a local address object.

`sockets:abstract-address-p` *object* [Generic Function]
Return `t` if `address` is a `local-address` that lives in the abstract namespace.

`sockets:address-type` *address* [Generic Function]
Returns a keyword symbol denoting the kind of `address` (:IPV4, :IPV6 or :LOCAL). If `address` is not a known address object, `nil` is returned.

`sockets:inet-address-loopback-p` *address* [Generic Function]
Returns `t` if `address` is a loopback internet address.

`sockets:inet-address-multicast-p` *address* [Generic Function]
Returns `t` if `address` is an multicast internet address.

`sockets:inet-address-type` *address* [Generic Function]
Returns the address type of `address` as 2 values:

- protocol, one of :IPV4 or :IPV6
- kind, one of `:unspecified`, `:loopback`, `:multicast` or `:unicast`

For unicast or multicast IPv6 addresses, a third value is returned which corresponds to the return value of IPV6-UNICAST-TYPE or IPV6-MULTICAST-TYPE, respectively.

`sockets:inet-address-unicast-p` *address* [Generic Function]
Returns `t` if `address` is an unicast internet address.

`sockets:inet-address-unspecified-p` *addr* [Generic Function]
Returns `t` if `addr` is an "unspecified" internet address.

`sockets:ipv6-admin-local-multicast-p` *address* [Function]
Returns `t` if `address` is a admin-local multicast IPv6 address.

`sockets:ipv6-global-multicast-p` *address* [Function]
Returns `t` if `address` is a global multicast IPv6 address.

`sockets:ipv6-global-unicast-p` *address* [Function]
Returns `t` if `address` is an global unicasst IPv6 address.

`sockets:ipv6-interface-local-multicast-p` *address* [Function]
Returns `t` if `address` is an interface-local IPv6 address.

`sockets:ipv6-ipv4-mapped-p` *address* [Function]
Returns `t` if `address` is an IPv6 address representing an IPv4 mapped address.

`sockets:ipv6-link-local-multicast-p` *address* [Function]
Returns `t` if `address` is a link-local IPv6 address.

`sockets:ipv6-link-local-unicast-p` *address* [Function]
Returns `t` if `address` is an link-local unicast IPv6 address.

`sockets:ipv6-multicast-type` *address* [Function]
Returns the multicast type of `address` or `nil` if it's not a multicast address.

`sockets:ipv6-organization-local-multicast-p` *address* [Function]
Returns `t` if `address` is an organization-local multicast IPv6 address.

`sockets:ipv6-reserved-multicast-p` *address* [Function]
Returns `t` if `address` is a reserved multicast IPv6 address.

`sockets:ipv6-site-local-multicast-p` *address* [Function]
Returns `t` if `address` is an site-local multicast IPv6 address.

`sockets:ipv6-site-local-unicast-p` *address* [Function]
Returns `t` if `address` is an site-local unicast IPv6 address.

`sockets:ipv6-solicited-node-multicast-p` *address* [Function]
Returns `t` if `address` is a solicited-node multicast IPv6 address.

`sockets:ipv6-transient-multicast-p` *address* [Function]
Returns `t` if `address` is a transient multicast IPv6 address.

`sockets:ipv6-unassigned-multicast-p` *address* [Function]
Returns `t` if `address` is an unassigned multicast IPv6 address.

`sockets:ipv6-unicast-type` *address* [Function]
Returns the unicast type of `address` or `nil` if it's not a unicast address.

2.3.3 Address Arithmetic

`sockets:make-netmask` **&key** *cidr class* [Function]
Create a subnet mask by specifying either its class(:A, `:b` or :C) or a `cidr` suffix(a number between 0 and 32).

`sockets:ensure-netmask` *thing* [Function]
If `thing` is of type IPV4-ADDRESS it is returned as is; if keyword it must be one of `:a`, `:b` or `:c` otherwise it's treated as a `cidr` suffix.

`sockets:inet-address-network-portion` *address netmask* [Generic Function]
Apply network netmask `netmask` to `address` in order to calculate the network part of `address`.

`sockets:inet-address-host-portion` *address netmask* [Generic Function]
Apply network netmask `netmask` to `address` in order to calculate the host part of `address`.

`sockets:inet-address-in-network-p` *address network* [Generic Function]
Return `t` if `address` is part of the subnet specified by `network`.

`sockets:inet-addresses-in-same-network-p` *address1 address2 network* [Generic Function]
Return `t` if ADDRESS1 and ADDRESS2 are both part part of the subnet specified by `network`.

`sockets:inet-address-network-class` *address* [Generic Function]
Return the network class of `address`: one of `:a`, `:b`, `:c`, `:d` or `:e` .

`sockets:inet-address-private-p` *address* [Generic Function]
Returns `t` if `address` is in a private network range. Private IPv4 networks are 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16. See http://en.wikipedia.org/wiki/Private_network for details.

2.3.4 Low-level Address Manipulation

`sockets:+max-ipv4-value+` [Constant]
Integer denoting 255.255.255.255

`sockets:dotted-to-integer` *address* [Function]
Convert a dotted IPv4 address to an (UNSIGNED-BYTE 32).

`sockets:integer-to-dotted` *integer* [Function]
Convert an (UNSIGNED-BYTE 32) IPv4 address to a dotted string.

`sockets:vector-to-integer` *vector* [Function]
Convert a vector to a 32-bit unsigned integer.

`sockets:integer-to-vector` *ipaddr* [Function]
Convert a 32-bit unsigned integer to a vector.

`sockets:dotted-to-vector` *address* [Function]
Convert a dotted IPv4 address to a (SIMPLE-ARRAY (UNSIGNED-BYTE 8) 4).

`sockets:vector-to-dotted` *vector* [Function]
Convert an 4-element vector to a dotted string.

`sockets:colon-separated-to-vector` *string* [Function]
Convert a colon-separated IPv6 address to a (SIMPLE-ARRAY (UNSIGNED-BYTE 16) 8).

`sockets:vector-to-colon-separated` *vector* **&optional** *case* [Function]
Convert an (SIMPLE-ARRAY (UNSIGNED-BYTE 16) 8) to a colon-separated IPv6 address. `case` may be `:downcase` or `:upcase`.

`sockets:string-address-to-vector` *address* [Function]
Convert a string address (dotted or colon-separated) to a vector address. If the string is not a valid address, return `nil`.

`sockets:map-ipv4-address-to-ipv6` *address* [Function]
Returns an IPv6 address by mapping `address` onto it.

`sockets:map-ipv6-address-to-ipv4` *address* [Function]
Extracts the IPv4 part of an IPv6-mapped IPv4 address. Signals an error if `address` is not an IPv6-mapped IPv4 address.

2.4 DNS

2.4.1 Functions

`sockets:lookup-hostname` *host* **&key** *ipv6* [Function]

Looks up a host by name or address. IPV6 determines the IPv6 behaviour, defaults to *IPV6*. Returns 4 values:

- an address
- a list of additional addresses(if existent)
- the canonical name of the host
- an alist of all the host's names with their respective addresses

`sockets:ensure-hostname` *address* **&key** *ipv6 errorp* [Function]

If `address` is an inet-address designator, it is converted, if necessary, to an `inet-address` object and returned. Otherwise it is assumed to be a host name which is then looked up in order to return its primary address as the first return value and the remaining address list as the second return value.

2.4.2 Conditions

`sockets:resolver-error` [Condition]

Class precedence list: `resolver-error, condition, t`

Signaled when an error occurs while trying to resolve an address.

`sockets:resolver-error-datum` *condition* [Generic Function]

Return the datum that caused the signalling of a `resolver-error` condition.

`sockets:resolver-again-error` [Condition]

Class precedence list: `resolver-again-error, resolver-error, condition, t`

Condition signaled when a temporary failure occurred.

`sockets:resolver-fail-error` [Condition]

Class precedence list: `resolver-fail-error, resolver-error, condition, t`

Condition signaled when a non-recoverable error occurred.

`sockets:resolver-no-name-error` [Condition]

Class precedence list: `resolver-no-name-error, resolver-error, condition, t`

Condition signaled when a host or service was not found.

`sockets:resolver-unknown-error` [Condition]

Class precedence list: `resolver-unknown-error, resolver-error, condition, t`

Condition signaled when an unknown error is signaled while resolving an address.

2.5 Services

`sockets:lookup-service` *service* **&optional** *protocol* [Function]
Lookup a service by port or name. `protocol` should be one of `:tcp`, `:udp` or `:any`.

`sockets:unknown-service` [Condition]
Class precedence list: `unknown-service, condition, t`

Condition raised when a network service is not found.

`sockets:unknown-service-datum` *condition* [Generic Function]
Return the datum that caused the signalling of an `unknown-service` condition.

2.6 Protocols

`sockets:lookup-protocol` *protocol* [Function]
Lookup a protocol by name or number. Signals an `unknown-protocol` error if no protocol is found.

`sockets:unknown-protocol` [Condition]
Class precedence list: `unknown-protocol, condition, t`

Condition raised when a network protocol is not found.

`sockets:unknown-protocol-datum` *condition* [Generic Function]
Return the datum that caused the signalling of an `unknown-protocol` condition.

2.7 Network Interfaces

`sockets:interface-index` *interface* [Function]
Return the `os` index of a network interface.

`sockets:interface-name` *interface* [Function]
Return the name of an network interface.

`sockets:make-interface` *name index* [Function]
Constructor for `interface` objects.

`sockets:lookup-interface` *interface* [Function]
Lookup an interface by name or index. `unknown-interface` is signalled if an interface is not found.

`sockets:list-network-interfaces` [Function]
Returns a list of network interfaces currently available.

`sockets:unknown-interface` [Condition]
Class precedence list: `unknown-interface, enxio, syscall-error, iolib-error, error, serious-condition, iolib-condition, condition, t`

Condition raised when a network interface is not found.

`sockets:unknown-interface-datum` *condition* [Generic Function]
Return the datum that caused the signalling of an `unknown-interface` condition.

Index