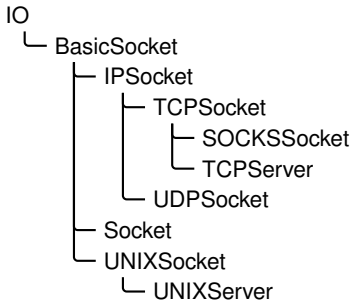
Appendix A

Socket Library

Because the socket and network libraries are such important parts of integrating Ruby applications with the 'net, we've decided to document them in more detail than the other standard libraries.

The hierarchy of socket classes is shown in the following diagram:



Because the socket calls are implemented in a library, you'll need to remember to add the following line to your code:

```
require 'socket'
```

Class **BasicSocket** < IO require "mkmf"

BasicSocket is an abstract base class for all other socket classes.

This class and its subclasses often manipulate addresses using something called a struct sockaddr, which is effectively an opaque binary string.[1]

Class methods

do_not_reverse_lookup BasicSocket.do_not_reverse_lookup → true or false

Returns the value of the global reverse lookup flag.

do_not_reverse_lookup= BasicSocket.do_not_reverse_lookup = true or false

Sets the global reverse lookup flag. If set to true, queries on remote addresses will return the numeric address but not the host name.

By default the socket library performs this reverse lookup on connections. If for some reason this lookup is slow or times out, connecting to a host can take a long time. Set this option to false to fix this.

for_fd BasicSocket.for_fd(*fd*) → *sock*

Wraps an already open file descriptor into a socket object.

Instance methods

close_read *sock*.close_read → nil

Closes the readable connection on this socket.

close_write *sock*.close_write → nil

Closes the writable connection on this socket.

getpeername *sock*.getpeername → *string*

Returns the struct sockaddr structure associated with the other end of this socket connection.

getsockname *sock*.getsockname → *string*

Returns the struct sockaddr structure associated with *sock*.

getsockopt *sock*.getsockopt(*level*, *optname*) → *string*

Returns the value of the specified option.

recv *sock*.recv(*len*, ⟨ , *flags* ⟩) → *string*

Receives up to *len* bytes from *sock*.

1. In reality, it maps onto the underlying C-language struct sockaddr set of structures, documented in the man pages and in the books by Stevens.

recv_nonblock *sock*.recv_nonblock(*len*, ⟨ , *flags* ⟩) → *string*

1.9 Receives up to *len* bytes from *sock* after first setting the socket into nonblocking mode. If the underlying recvfrom call returns 0, an empty string is returned.

send *sock*.send(*string*, *flags*, ⟨ , *to* ⟩) → *int*

Sends *string* over *sock*. If specified, *to* is a struct sockaddr specifying the recipient address. *flags* are the sum of one or more of the MSG_ options (listed on the next page). Returns the number of characters sent.

setsockopt *sock*.setsockopt(*level*, *optname*, *optval*) → 0

Sets a socket option. *level* is one of the socket-level options (listed on the following page). *optname* and *optval* are protocol specific—see your system documentation for details.

shutdown *sock*.shutdown(*how*=2) → 0

Shuts down the receive (*how* == 0), sender (*how* == 1), or both (*how* == 2), parts of this socket.

Class **Socket** < BasicSocket require "mkmf"

Class Socket provides access to the operating system socket implementation. It can be used to provide more system–specific functionality than the protocol-specific socket classes but at the expense of greater complexity. In particular, the class handles addresses using struct sockaddr structures packed into Ruby strings, which can be a joy to manipulate.

Class constants

Constants are available only on architectures that support the related facility.

Types:
SOCK_DGRAM, SOCK_PACKET, SOCK_RAW, SOCK_RDM, SOCK_SEQPACKET, SOCK_STREAM

Protocol families:
PF_APPLETALK, PF_AX25, PF_INET6, PF_INET, PF_IPX, PF_UNIX, PF_UNSPEC

Address families:
AF_APPLETALK, AF_AX25, AF_INET6, AF_INET, AF_IPX, AF_UNIX, AF_UNSPEC

Lookup-order options:
LOOKUP_INET6, LOOKUP_INET, LOOKUP_UNSPEC

Send/receive options:
MSG_DONTROUTE, MSG_OOB, MSG_PEEK

Socket-level options:
SOL_ATALK, SOL_AX25, SOL_IPX, SOL_IP, SOL_SOCKET, SOL_TCP, SOL_UDP

Socket options:
SO_BROADCAST, SO_DEBUG, SO_DONTROUTE, SO_ERROR, SO_KEEPALIVE, SO_LINGER, SO_NO_CHECK, SO_OOBINLINE, SO_PRIORITY, SO_RCVBUF, SO_REUSEADDR, SO_SNDBUF, SO_TYPE

QOS options:
SOPRI_BACKGROUND, SOPRI_INTERACTIVE, SOPRI_NORMAL

Multicast options:
IP_ADD_MEMBERSHIP, IP_DEFAULT_MULTICAST_LOOP, IP_DEFAULT_MULTICAST_TTL, IP_MAX_MEMBERSHIPS, IP_MULTICAST_IF, IP_MULTICAST_LOOP, IP_MULTICAST_TTL

TCP options:
TCP_MAXSEG, TCP_NODELAY

getaddrinfo error codes:
EAI_ADDRFAMILY, EAI_AGAIN, EAI_BADFLAGS, EAI_BADHINTS, EAI_FAIL, EAI_FAMILY, EAI_MAX, EAI_MEMORY, EAI_NODATA, EAI_NONAME, EAI_PROTOCOL, EAI_SERVICE, EAI_SOCKTYPE, EAI_SYSTEM

ai_flags values:
AI_ALL, AI_CANONNAME, AI_MASK, AI_NUMERICHOST, AI_PASSIVE, AI_V4MAPPED_CFG

Class methods

getaddrinfo

Socket.getaddrinfo(*hostname*, *port*, ⟨ *family* ⟨ , *socktype* ⟨ , *protocol* ⟨ , *flags* ⟩ ⟩ ⟩ ⟩) → *array*

Returns an array of arrays describing the given host and port (optionally qualified as shown). Each subarray contains the address family, port number, host name, host IP address, protocol family, socket type, and protocol.

```
require 'socket'
  for line in Socket.getaddrinfo('www.microsoft.com', 'http')
    puts line.join(", ")
  end
```

produces:

```
AF_INET, 80, wwwbaytest1.microsoft.com, 207.46.19.190, 2, 2, 17
AF_INET, 80, wwwbaytest1.microsoft.com, 207.46.19.190, 2, 1, 6
AF_INET, 80, wwwbaytest2.microsoft.com, 207.46.19.254, 2, 2, 17
AF_INET, 80, wwwbaytest2.microsoft.com, 207.46.19.254, 2, 1, 6
```

gethostbyaddr

Socket.gethostbyaddr(*addr*, *type*=AF_INET) → *array*

Returns the host name, address family, and sockaddr component for the given address.

```
a = Socket.gethostbyname("198.145.243.54")
res = Socket.gethostbyaddr(a[3], a[2])
res.join(', ')   # =>   "mike.pragprog.com, , 2, \xC6\x91\xF36"
```

gethostbyname

Socket.gethostbyname(*hostname*) → *array*

Returns a four-element array containing the canonical host name, a subarray of host aliases, the address family, and the address portion of the sockaddr structure.

```
a = Socket.gethostbyname("63.68.129.130")
a.join(', ')   # =>   "63.68.129.130, , 2, ?D\x81\x82"
```

gethostname

Socket.gethostname → *string*

Returns the name of the current host.

```
Socket.gethostname   # =>   "dave-2.home"
```

getnameinfo

Socket.getnameinfo(*addr* ⟨ , *flags* ⟩) → *array*

Looks up the given address, which may be either a string containing a sockaddr or a three- or four-element array. If *addr* is an array, it should contain the string address family, the port (or nil), and the host name or IP address. If a fourth element is present and not nil, it will be used as the host name. Returns a canonical host name (or address) and port number as an array.

```
Socket.getnameinfo(["AF_INET", '23', 'www.ruby-lang.org'])
```

getservbyname

Socket.getservbyname(*service*, *proto*='tcp') → *int*

Returns the port corresponding to the given service and protocol.

```
Socket.getservbyname("telnet")   # =>   23
```

getservbyport Socket.getservbyport(*port*, *proto*='tcp') → *string*

1.9 Returns the port corresponding to the given service and protocol.

```
Socket.getservbyport(23)   # =>   "telnet"
```

new Socket.new(*domain*, *type*, *protocol*) → *sock*

Creates a socket using the given parameters.

open Socket.open(*domain*, *type*, *protocol*) → *sock*

Synonym for Socket.new.

pack_sockaddr_in Socket.pack_sockaddr_in(port, host) → *str_address*

Given a port and a host, returns the (system dependent) sockaddr structure as a string of bytes.

```
require 'socket'
addr = Socket.pack_sockaddr_in(80, "pragprog.com")
# Pragprog.com is 65.74.171.137
addr.unpack("CCnC4")   # =>   [16, 2, 80, 65, 74, 171, 137]
```

pack_sockaddr_un Socket.pack_sockaddr_un(path) → *str_address*

Given a path to a Unix socket, returns the (system dependent) sock_addr_un structure as a string of bytes. Available only on boxes supporting the Unix address family.

```
require 'socket'
addr = Socket.pack_sockaddr_un("/tmp/sample")
addr[0,20]   # =>   "\x00\x01/tmp/sample\x00\x00\x00\x00\x00\x00\x00\x00"
```

pair Socket.pair(*domain*, *type*, *protocol*) → *array*

Returns an array containing a pair of connected, anonymous Socket objects with the given domain, type, and protocol.

socketpair Socket.socketpair(*domain*, *type*, *protocol*) → *array*

Synonym for Socket.pair.

sockaddr_in Socket.sockaddr_in(port, host) → *str_address*

1.9 Synonym for pack_sockaddr_in.

sockaddr_un Socket.sockaddr_un(path) → *str_address*

1.9 Synonym for pack_sockaddr_un.

socket_pair Socket.socket_pair(*domain*, *type*, *protocol*) → *array*

Synonym for pair.

unpack_sockaddr_in Socket.pack_sockaddr_in(string_address) → [*port*, *host*]

Given a string containing a binary addrinfo structure, return the port and host.

```
require 'socket'
addr = Socket.pack_sockaddr_in(80, "pragprog.com")
Socket.unpack_sockaddr_in(addr)   # =>   [80, "65.74.171.137"]
```

unpack_sockaddr_un Socket.pack_sockaddr_in(string_address) → [*port*, *host*]

Given a string containing a binary sock_addr_un structure, returns the path to the Unix socket. Available only on boxes supporting the Unix address family.

```
require 'socket'
addr = Socket.pack_sockaddr_in(80, "pragprog.com")
Socket.unpack_sockaddr_in(addr)   # =>   [80, "65.74.171.137"]
```

Instance methods

accept *sock*.accept → [*socket*, *address*]

Accepts an incoming connection returning an array containing a new Socket object and a string holding the struct sockaddr information about the caller.

accept_nonblock *sock*.accept_nonblock → [*socket*, *address*]

1.9

Puts the listening socket into nonblocking mode and then accepts an incoming connection. Throws an exception if no connection is pending. You'll probably use this in conjunction with select.

bind *sock*.bind(*sockaddr*) → 0

Binds to the given struct sockaddr, contained in a string.

connect *sock*.connect(*sockaddr*) → 0

Connects to the given struct sockaddr, contained in a string.

listen *sock*.listen(*int*) → 0

Listens for connections, using the specified *int* as the backlog.

recvfrom *sock*.recvfrom(*len* ⟨ , *flags* ⟩) → [*data*, *sender*]

Receives up to *len* bytes from *sock*. *flags* is zero or more of the MSG_ options. The first element of the result is the data received. The second element contains protocol-specific information on the sender.

recvfrom_nonblock *sock*.recvfrom_nonblock(*len* ⟨ , *flags* ⟩) → [*data*, *sender*]

1.9

Receives up to *len* bytes from *sock* in nonblocking mode. *flags* is zero or more of the MSG_ options. The first element of the result is the data received. The second element contains protocol-specific information on the sender.

sysaccept *sock*.sysaccept → [*socket_fd*, *address*]

Accepts an incoming connection. Returns an array containing the (integer) file descriptor of the incoming connection and a string holding the struct sockaddr information about the caller.

Class **IPSocket** < BasicSocket require "mkmf"

Class IPSocket is a base class for sockets using IP as their transport. TCPSocket and UDPSocket are based on this class.

Class methods

getaddress

IPSocket.getaddress(*hostname*) → *string*

Returns the dotted-quad IP address of *hostname*.

```
a = IPSocket.getaddress('www.ruby-lang.org')
a   # =>   "221.186.184.68"
```

Instance methods

addr

sock.addr → *array*

Returns the domain, port, name, and IP address of *sock* as a four-element array. The name will be returned as an address if the do_not_reverse_lookup flag is true.

```
u = UDPSocket.new
u.bind('localhost', 8765)
u.addr   # =>   ["AF_INET", 8765, "localhost", "127.0.0.1"]
BasicSocket.do_not_reverse_lookup = true
u.addr   # =>   ["AF_INET", 8765, "localhost", "127.0.0.1"]
```

peeraddr

sock.peeraddr → *array*

Returns the domain, port, name, and IP address of the peer.

recvfrom

sock.recvfrom(*len* ⟨ , *flags* ⟩) → [*data*, *sender*]

Receives up to *len* bytes on the connection. *flags* is zero or more of the MSG_ options (listed on page 881). Returns a two-element array. The first element is the received data, and the second is an array containing information about the peer. On systems such as my Mac OS X box where the native recvfrom() method does not return peer information for TCP connections, the second element of the array is nil.

```
require 'socket'
t = TCPSocket.new('127.0.0.1', 'ftp')
data = t.recvfrom(40)
data      # =>   ["220 localhost FTP server (tnftpd 2006121", nil]
t.close   # =>   nil
```

Class **TCPSocket** < IPSocket require "mkmf"

```
t = TCPSocket.new('localhost', 'ftp')
t.gets    # =>   "220 localhost FTP server (tnftpd 20061217) ready.\r\n"
t.close   # =>   nil
```

Class methods

gethostbyname

TCPSocket.gethostbyname(*hostname*) → *array*

Looks up *hostname* and returns its canonical name, an array containing any aliases, the address type (AF_INET), and the dotted-quad IP address.

```
a = TCPSocket.gethostbyname('ns.pragprog.com')
a   # =>   ["pragprog.com", ["ns.pragprog.com"], 2, "65.74.171.137"]
```

new

TCPSocket.new(*hostname*, *port*) → *sock*

Opens a TCP connection to *hostname* on the *port*.

open

TCPSocket.open(*hostname*, *port*) → *sock*

Synonym for TCPSocket.new.

Class **SOCKSSocket** < TCPSocket require "mkmf"

Class SOCKSSocket supports connections based on the SOCKS protocol.

Class methods

new SOCKSSocket.new(*hostname*, *port*) → *sock*

Opens a SOCKS connection to *port* on *hostname*.

open SOCKSSocket.open(*hostname*, *port*) → *sock*

Synonym for SOCKSSocket.new.

Instance methods

close *sock*.close → nil

Closes this SOCKS connection.

Class **TCPServer** < TCPSocket require "mkmf"

A TCPServer accepts incoming TCP connections. Here is a web server that listens on a given port and returns the time:

```
require 'socket'
port = (ARGV[0] || 80).to_i
server = TCPServer.new('localhost', port)
while (session = server.accept)
  puts "Request: #{session.gets}"
  session.print "HTTP/1.1 200/OK\r\nContent-type: text/html\r\n\r\n"
  session.print "<html><body><h1>#{Time.now}</h1></body></html>\r\n"
  session.close
end
```

Class methods

new TCPServer.new(⟨ *hostname*, ⟩ *port*) → *sock*

Creates a new socket on the given interface (identified by *hostname* and port). If *hostname* is omitted, the server will listen on all interfaces on the current host (equivalent to an address of 0.0.0.0).

open TCPServer.open(⟨ *hostname*, ⟩ *port*) → *sock*

Synonym for TCPServer.new.

Instance methods

accept *sock*.accept → *tcp_socket*

Waits for a connection on *sock*and returns a new tcp_socket connected to the caller. See the example on this page.

Class **UDPSocket** < IPSocket require "mkmf"

UDP sockets send and receive datagrams. To receive data, a socket must be bound to a particular port. You have two choices when sending data: you can connect to a remote UDP socket and thereafter send datagrams to that port, or you can specify a host and port every time you send a packet. The following example is a UDP server that prints the message it receives. It is called by both connectionless and connection-based clients.

```
require 'socket'
PORT = 4321
server = UDPSocket.open
server.bind(nil, PORT)
server_thread = Thread.start(server) do |server|  # run server in a thread
  3.times { p server.recvfrom(64) }
end

# Ad-hoc client
UDPSocket.open.send("ad hoc", 0, 'localhost', PORT)
# Connection based client
sock = UDPSocket.open
sock.connect('localhost', PORT)
sock.send("connection-based", 0)
sock.send("second message", 0)
server_thread.join
```

produces:

```
["ad hoc", ["AF_INET", 55732, "localhost", "127.0.0.1"]]
["connection-based", ["AF_INET", 55733, "localhost", "127.0.0.1"]]
["second message", ["AF_INET", 55733, "localhost", "127.0.0.1"]]
```

Class methods

new UDPSocket.new(*family* = AF_INET) → *sock*

Creates a UDP endpoint, optionally specifying an address family.

open UDPSocket.open(*family* = AF_INET) → *sock*

Synonym for UDPSocket.new.

Instance methods

bind *sock*.bind(*hostname*, *port*) → 0

Associates the local end of the UDP connection with a given *hostname* and *port*. As well as a host name, the first parameter may be "<broadcast>" or "" (the empty string) to bind to INADDR_BROADCAST and INADDR_ANY, respectively. Must be used by servers to establish an accessible endpoint.

connect *sock*.connect(*hostname*, *port*) → 0

Creates a connection to the given *hostname* and *port*. Subsequent UDPSocket#send requests that don't override the recipient will use this connection. Multiple connect requests may be issued on *sock*: the most recent will be used by send. As well as a host name, the first parameter may be "<broadcast>" or "" (the empty string) to bind to INADDR_BROADCAST and INADDR_ANY, respectively.

recvfrom *sock*.recvfrom(*len* ⟨ , *flags* ⟩) → [*data*, *sender*]

Receives up to *len* bytes from *sock*. *flags* is zero or more of the MSG_ options (listed on page 881). The result is a two-element array containing the received data and information on the sender. See the example on the preceding page.

recvfrom_nonblock *sock*.recvfrom_nonblock(*len* ⟨ , *flags* ⟩) → [*data*, *sender*]

1.9

Receives up to *len* bytes from *sock* in nonblocking mode.

send *sock*.send(*string*, *flags*) → *int*
sock.send(*string*, *flags*, *hostname*, *port*) → *int*

The two-parameter form sends *string* on an existing connection. The four-parameter form sends *string* to *port* on *hostname*.

Class **UNIXSocket** < BasicSocket require "mkmf"

Class UNIXSocket supports interprocess communications using the Unix domain protocol. Although the underlying protocol supports both datagram and stream connections, the Ruby library provides only a stream-based connection.

```
require 'socket'
SOCKET = "/tmp/sample"
sock = UNIXServer.open(SOCKET)
server_thread = Thread.start(sock) do |sock|      # run server in a thread
  s1 = sock.accept
  p s1.recvfrom(124)
end
client = UNIXSocket.open(SOCKET)
client.send("hello", 0)
client.close
server_thread.join
```

produces:

```
["hello", ["AF_UNIX", ""]]
```

Class methods

new UNIXSocket.new(*path*) → *sock*

Opens a new domain socket on *path*, which must be a path name.

open UNIXSocket.open(*path*) → *sock*

Synonym for UNIXSocket.new.

Instance methods

addr *sock*.addr → *array*

Returns the address family and path of this socket.

path *sock*.path → *string*

Returns the path of this domain socket.

peeraddr *sock*.peeraddr → *array*

Returns the address family and path of the server end of the connection.

recvfrom *sock*.recvfrom(*len* ⟨ , *flags* ⟩) → *array*

Receives up to *len* bytes from *sock*. *flags* is zero or more of the MSG_ options (listed on page 881). The first element of the returned array is the received data, and the second contains (minimal) information on the sender.

Class **UNIXServer** < UNIXSocket require "mkmf"

Class UNIXServer provides a simple Unix domain socket server. See UNIXSocket for example code.

Class methods

new UNIXServer.new(*path*) → *sock*

Creates a server on the given *path*. The corresponding file must not exist at the time of the call.

open UNIXServer.open(*path*) → *sock*

Synonym for UNIXServer.new.

Instance methods

accept *sock*.accept → *unix_socket*

Waits for a connection on the server socket and returns a new socket object for that connection. See the example for UNIXSocket on the preceding page.