For our work we may use AF\_INET socket family that have a pair (host, port)

Socket module funcs

socket.socket([*family*[, *type*[, *proto*]]])

Create a new socket using the given address family, socket type and protocol number. The address family should be [AF\_INET](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.AF_INET) (the default), [AF\_INET6](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.AF_INET6), [AF\_UNIX](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.AF_UNIX), [AF\_CAN](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.AF_CAN) or [AF\_RDS](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.AF_RDS). The socket type should be [SOCK\_STREAM](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.SOCK_STREAM) (the default), [SOCK\_DGRAM](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.SOCK_DGRAM), [SOCK\_RAW](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.SOCK_RAW) or perhaps one of the other SOCK\_ constants. The protocol number is usually zero and may be omitted in that case or CAN\_RAW in case the address family is [AF\_CAN](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.AF_CAN).

*exception* socket.error

A deprecated alias of [OSError](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/exceptions.html#OSError).

*exception* socket.timeout

A subclass of [OSError](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/exceptions.html#OSError), this exception is raised when a timeout occurs on a socket which has had timeouts enabled via a prior call to [settimeout()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.settimeout) (or implicitly through [setdefaulttimeout()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.setdefaulttimeout)). The accompanying value is a string whose value is currently always “timed out”.

socket.has\_ipv6

This constant contains a boolean value which indicates if IPv6 is supported on this platform.

socket.create\_connection(*address*[, *timeout*[, *source\_address*]])

Connect to a TCP service listening on the Internet *address* (a 2-tuple (host, port)), and return the socket object. This is a higher-level function than [socket.connect()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.connect): if *host* is a non-numeric hostname, it will try to resolve it for both [AF\_INET](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.AF_INET) and [AF\_INET6](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.AF_INET6), and then try to connect to all possible addresses in turn until a connection succeeds. This makes it easy to write clients that are compatible to both IPv4 and IPv6.

Passing the optional *timeout* parameter will set the timeout on the socket instance before attempting to connect. If no *timeout* is supplied, the global default timeout setting returned by [getdefaulttimeout()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.getdefaulttimeout) is used.

socket.getaddrinfo(*host*, *port*, *family=0*, *type=0*, *proto=0*, *flags=0*)

Translate the *host*/*port* argument into a sequence of 5-tuples that contain all the necessary arguments for creating a socket connected to that service. *host* is a domain name, a string representation of an IPv4/v6 address or None. *port* is a string service name such as 'http', a numeric port number or None. By passing None as the value of *host* and *port*, you can pass NULL to the underlying C API.

The *family*, *type* and *proto* arguments can be optionally specified in order to narrow the list of addresses returned. Passing zero as a value for each of these arguments selects the full range of results. The *flags* argument can be one or several of the AI\_\* constants, and will influence how results are computed and returned. For example, AI\_NUMERICHOST will disable domain name resolution and will raise an error if *host* is a domain name.

The function returns a list of 5-tuples with the following structure:

(family, type, proto, canonname, sockaddr)

socket.getfqdn([*name*])

Return a fully qualified domain name for *name*. If *name* is omitted or empty, it is interpreted as the local host. To find the fully qualified name, the hostname returned by [gethostbyaddr()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.gethostbyaddr) is checked, followed by aliases for the host, if available. The first name which includes a period is selected. In case no fully qualified domain name is available, the hostname as returned by [gethostname()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.gethostname) is returned.

socket.gethostbyname(*hostname*)

Translate a host name to IPv4 address format. The IPv4 address is returned as a string, such as '100.50.200.5'. If the host name is an IPv4 address itself it is returned unchanged. See [gethostbyname\_ex()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.gethostbyname_ex) for a more complete interface. [gethostbyname()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.gethostbyname) does not support IPv6 name resolution, and [getaddrinfo()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.getaddrinfo) should be used instead for IPv4/v6 dual stack support.

socket.gethostname()

Return a string containing the hostname of the machine where the Python interpreter is currently executing.

If you want to know the current machine’s IP address, you may want to use gethostbyname(gethostname()). This operation assumes that there is a valid address-to-host mapping for the host, and the assumption does not always hold.

socket.getnameinfo(*sockaddr*, *flags*)

Translate a socket address *sockaddr* into a 2-tuple (host, port). Depending on the settings of *flags*, the result can contain a fully-qualified domain name or numeric address representation in *host*. Similarly, *port* can contain a string port name or a numeric port number.

socket.getdefaulttimeout()

Return the default timeout in seconds (float) for new socket objects. A value of None indicates that new socket objects have no timeout. When the socket module is first imported, the default is None.

socket.setdefaulttimeout(*timeout*)

Set the default timeout in seconds (float) for new socket objects. When the socket module is first imported, the default is None. See [settimeout()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.settimeout) for possible values and their respective meanings.

socket.sethostname(*name*)

Set the machine’s hostname to *name*. This will raise a [OSError](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/exceptions.html#OSError) if you don’t have enough rights.

socket.fromshare(*data*)

Instantiate a socket from data obtained from [share()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.share). The socket is assumed to be in blocking mode.

Socket object funcs

socket.bind(*address*)

Bind the socket to *address*. The socket must not already be bound. (The format of *address* depends on the address family — see above.)

socket.accept()

Accept a connection. The socket must be bound to an address and listening for connections. The return value is a pair (conn, address) where *conn* is a *new* socket object usable to send and receive data on the connection, and *address* is the address bound to the socket on the other end of the connection.

socket.close()

Close the socket. All future operations on the socket object will fail. The remote end will receive no more data (after queued data is flushed). Sockets are automatically closed when they are garbage-collected.

[close()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.close) releases the resource associated with a connection but does not necessarily close the connection immediately. If you want to close the connection in a timely fashion, call [shutdown()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.shutdown) before [close()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.close).

socket.connect(*address*)

Connect to a remote socket at *address*. (The format of *address* depends on the address family — see above.)

socket.fileno()

Return the socket’s file descriptor (a small integer). This is useful with [select.select()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/select.html#select.select).

socket.getpeername()

Return the remote address to which the socket is connected. This is useful to find out the port number of a remote IPv4/v6 socket, for instance. (The format of the address returned depends on the address family — see above.) On some systems this function is not supported.

socket.getsockname()

Return the socket’s own address. This is useful to find out the port number of an IPv4/v6 socket, for instance. (The format of the address returned depends on the address family — see above.)

socket.gettimeout()

Return the timeout in seconds (float) associated with socket operations, or None if no timeout is set. This reflects the last call to [setblocking()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.setblocking) or [settimeout()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.settimeout).

socket.listen(*backlog*)

Listen for connections made to the socket. The *backlog* argument specifies the maximum number of queued connections and should be at least 0; the maximum value is system-dependent (usually 5), the minimum value is forced to 0.

socket.makefile(*mode='r'*, *buffering=None*, *\**, *encoding=None*, *errors=None*, *newline=None*)

Return a [*file object*](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/glossary.html#term-file-object) associated with the socket. The exact returned type depends on the arguments given to [makefile()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.makefile). These arguments are interpreted the same way as by the built-in [open()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/functions.html#open) function.

socket.recv(*bufsize*[, *flags*])

Receive data from the socket. The return value is a bytes object representing the data received. The maximum amount of data to be received at once is specified by *bufsize*. See the Unix manual page *recv(2)* for the meaning of the optional argument *flags*; it defaults to zero.

Note

For best match with hardware and network realities, the value of *bufsize* should be a relatively small power of 2, for example, 4096.

socket.recvfrom(*bufsize*[, *flags*])

Receive data from the socket. The return value is a pair (bytes, address) where *bytes* is a bytes object representing the data received and *address* is the address of the socket sending the data. See the Unix manual page *recv(2)* for the meaning of the optional argument *flags*; it defaults to zero. (The format of *address* depends on the address family — see above.)

socket.recvmsg(*bufsize*[, *ancbufsize*[, *flags*]])

Receive normal data (up to *bufsize* bytes) and ancillary data from the socket. The *ancbufsize* argument sets the size in bytes of the internal buffer used to receive the ancillary data; it defaults to 0, meaning that no ancillary data will be received. Appropriate buffer sizes for ancillary data can be calculated using [CMSG\_SPACE()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.CMSG_SPACE) or [CMSG\_LEN()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.CMSG_LEN), and items which do not fit into the buffer might be truncated or discarded. The *flags* argument defaults to 0 and has the same meaning as for [recv()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.recv).

The return value is a 4-tuple: (data, ancdata, msg\_flags, address). The *data* item is a [bytes](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/functions.html#bytes) object holding the non-ancillary data received. The *ancdata* item is a list of zero or more tuples (cmsg\_level, cmsg\_type, cmsg\_data) representing the ancillary data (control messages) received: *cmsg\_level* and *cmsg\_type* are integers specifying the protocol level and protocol-specific type respectively, and *cmsg\_data* is a [bytes](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/functions.html#bytes) object holding the associated data. The *msg\_flags* item is the bitwise OR of various flags indicating conditions on the received message; see your system documentation for details. If the receiving socket is unconnected, *address* is the address of the sending socket, if available; otherwise, its value is unspecified.

socket.send(*bytes*[, *flags*])

Send data to the socket. The socket must be connected to a remote socket. The optional *flags* argument has the same meaning as for [recv()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.recv) above. Returns the number of bytes sent. Applications are responsible for checking that all data has been sent; if only some of the data was transmitted, the application needs to attempt delivery of the remaining data. For further information on this topic, consult the [*Socket Programming HOWTO*](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/howto/sockets.html#socket-howto).

socket.sendall(*bytes*[, *flags*])

Send data to the socket. The socket must be connected to a remote socket. The optional *flags* argument has the same meaning as for [recv()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.recv) above. Unlike [send()](mk:@MSITStore:C:\Python33\Doc\Python330.chm::/library/socket.html#socket.socket.send), this method continues to send data from *bytes* until either all data has been sent or an error occurs. None is returned on success. On error, an exception is raised, and there is no way to determine how much data, if any, was successfully sent.

Socket consts

For socket family use : socket.AF\_INET

& for TCP protocol use : socket.SOCK\_STREAM

For UDP protocol socket.SOCK\_DGRAM