

Definition: WinSock is the standard sockets programming [API](#) for the Windows operating system. WinSock has been the standard sockets library shipped with all versions of Windows starting with Windows 95.

WinSock was created to allow different Microsoft Windows [TCP/IP](#) software applications to communicate. WinSock borrowed and expanded on the concept of [sockets](#) and socket programming first made popular on Unix computer systems in the 1980s. WinSock most closely matches the Berkeley implementation of Unix sockets.

Two major versions of WinSock exist for Windows. All implementations of WinSock are packaged in a single Windows dynamic-link library (DLL). The current version of Windows WinSock, version 2.2, is contained in the **WS2_32.DLL** library. Older Winsock version 1 libraries are named either **WINSOCK.DLL** or **WSOCK32.DLL**. Newer releases of Windows WinSock have retained backward compatibility with the older WinSock versions.

A socket is one of the most fundamental technologies of computer networking. Sockets allow applications to communicate using standard mechanisms built into network hardware and operating systems. Although network software may seem to be a relatively new "Web" phenomenon, socket technology actually has been employed for roughly two decades.

Software applications that rely on the Internet and other computer networks continue to grow in popularity. Many of today's most popular software packages -- including Web browsers, instant messaging applications and peer to peer file sharing systems -- rely on sockets.

Point-to-Point Communication

In a nutshell, a socket represents a single connection between exactly two pieces of software. More than two pieces of software can communicate in *client/server* or *distributed* systems (for example, many Web browsers can simultaneously communicate with a single Web server) but multiple sockets are required to do this. Socket-based software usually runs on two separate computers on the network, but sockets can also be used to communicate locally (*interprocess*) on a single computer.

Sockets are *bidirectional*, meaning that either side of the connection is capable of both sending and receiving data. Sometimes the one application that initiates communication is termed the *client* and the other application the *server*, but this terminology leads to confusion in non-client/server systems and should generally be avoided.

Libraries

Programmers access sockets using code libraries packaged with the operating system. Several libraries that implement standard *application programming interfaces* (APIs) exist. The first mainstream package - the Berkeley Socket Library is

still widely in use on UNIX® systems. Another very common API is the [Windows Sockets \(Winsock\)](#) library for Microsoft operating systems. Relative to other network programming technologies, socket APIs are quite mature: Winsock has been in use since 1993 and Berkeley sockets since 1982.

Interface Types

Socket interfaces can be divided into three categories. Perhaps the most commonly-used type, the *stream* socket, implements "connection-oriented" semantics. Essentially, a "stream" requires that the two communicating parties first establish a socket connection, after which any data passed through that connection will be guaranteed to arrive in the same order in which it was sent.

Datagram sockets offer "connection-less" semantics. With datagrams, connections are implicit rather than explicit as with streams. Either party simply sends datagrams as needed and waits for the other to respond; messages can be lost in transmission or received out of order, but it is the application's responsibility and not the socket's to deal with these problems. Implementing datagram sockets can give some applications a performance boost and additional flexibility compared to using stream sockets, justifying their use in some situations.

The third type of socket -- the so-called *raw* socket -- bypasses the library's built-in support for standard protocols like TCP and UDP. Raw sockets are used for custom low-level protocol development.

Addresses and Ports

Today, sockets are typically used in conjunction with the Internet protocols -- Internet Protocol, Transmission Control Protocol, and [User Datagram Protocol \(UDP\)](#). Libraries implementing sockets for Internet Protocol use TCP for streams, UDP for datagrams, and IP itself for raw sockets.

To communicate over the Internet, IP socket libraries use the IP [address](#) to identify specific computers. Many parts of the Internet work with *naming services*, so that the users and socket programmers can work with computers by name (*e.g.*, "thiscomputer.compnetworking.about.com") instead of by address (*e.g.*, 208.185.127.40). Stream and datagram sockets also use IP [port numbers](#) to distinguish multiple applications from each other. For example, Web browsers on the Internet know to use port 80 as the default for socket communications with Web servers.

Socket Programming and You

Traditionally, sockets have been of interest mainly to computer programmers. But as new networking applications emerge, end users are becoming increasingly network-savvy. Many Web surfers, for example, now know that some addresses in the browser look like

`http://206.35.113.28:8080/`

where 8080 is the port number being used by that socket.

The socket APIs are relatively small and simple. Many of the functions are similar to those used in file input/output routines such as `read()`, `write()`, and `close()`. The actual function calls to use depend on the programming language and socket library chosen.