A "Java-Like" Socket Wrapper Class for TCP/IP Communications under Win32

Tom Wheeler, NØGSG
twheeler@kc.devry.edu

Introduction

The Win32 sockets API is based on BSD sockets and is fairly complex to program. It provides most services programmers would likely need. Java, a newer language than C++, provides a powerful set of classes for network communications that are much easier to use than the Windows sockets calls. So why not create a set of wrapper classes to simplify Windows TCP/IP programming so that it can be done in the Java style? Java sockets are much easier to understand and manipulate, and the resulting code is much more concise. That is exactly what this wrapper class accomplishes. Using these classes, UDP and TCP communications can be achieved in just a few lines of code, even for multi-threaded applications (these classes are thread-safe.)

What the Classes Don't Do (Limitations)

There are certain features that I have not chosen to incorporate into these wrapper classes. This is many due to time constraints. Here's what's been left out:

- No Structured Exception Handling (SEH). After performing an I/O operation, no exceptions are thrown by these classes. Generally, a result code is returned that lets you know what happened.

- Single Host DNS Only. The InetAddress class only resolves to a single target IP address. The first found IP address is returned. If more IPs exist for the domain, this class does not yet return them.

- DNS Query only for (A)ddress. The InetAddress class can't query for different types of IP data (such as MX records, etc.) This capability will be added to a future version of the InetAddress class.

- IP Version 4 Only. These classes are functional only under IPv4. IPv6 support will be added at a later time.

- Methods not 100% Java Equivalent. Read the source code documentation carefully; these classes avoid the Java streams model entirely, and many methods use a slightly different name. However, that being said, I/O using these sockets is much easier than Java I/O since direct reads and writes to the sockets are permitted. (For example, to transmit anything other than an array of bytes by UDP in Java sockets, you must manipulate the data into a byte array by using other classes to help. In this implementation, you only have to tell the UDP socket to transmit the data, with no transformations needed.)

Summary of the Classes

`Socket()`

The Socket class performs TCP point-to-point (unicast) communications. By constructing a Socket object, a client connection to a server can be established. For example, the statement

```
Socket s("yahoo.com", 80);
```

constructs a socket *s* with a connection to host *`yahoo.com`* on destination port 80. (To see if the connect was successful, use the `GetStatus()` method of `Socket` after doing the connection attempt.)

`ServerSocket()`

The `ServerSocket` class is designed to accept incoming TCP connections. If a number is supplied to the `ServerSocket` constructor, the socket is automatically bound to the specified port number.

Calling the `Accept()` method on the `ServerSocket` object causes the socket to wait for an incoming TCP connection request from a client. On success, `Accept()` returns a pointer to a newly-constructed `Socket` object for communications with the client.

`ServerSocket` objects can also be manually bound to a port by calling `Bind()`, and can be manually unbound by calling `Close()`. This will allow software to dynamically bind and unbind server sockets without the requirement of object destruction.

`DatagramSocket()`

The DatagramSocket class encapsulates UDP communications. Broadcasting is always supported and enabled. DatagramSockets are connectionless and are constructed either as server type (a port number is supplied to the constructor, and the socket can only receive data) or client type (no arguments are passed to the constructor, and the socket can only send data).

`InetAddress()`

The `InetAddress` class encapsulates raw IP addresses. This class can be used for manual DNS resolution of hosts, or just for storage of IP addresses (for example, a server-type `DatagramSocket` can be asked where it got its last packet from by calling the `GetSenderAddress()` method, which returns an `InetAddress` object.) An `InetAddress` object can be used in place of a hostname when constructing `Socket` objects.

Example Code

An example use of all the wrapper classes is in the archive "socket.zip" located at http://faculty.kc.devry.edu/twheeler/projects/socket.zip. This is a short C++ program that demonstrates the TCP and UDP capabilities of the classes.

The demonstration program is a Visual C 6.0 project. To use it, unzip the contents into an empty folder, then open the project by double-clicking the ".dsw" file. If you're using .NET, you'll need to allow the framework to convert the project into .NET format.

The demo code tests the TCP client capabilities by opening a connection to a web server of your choice. It then loads the HTML web page and displays the HTML source code on the console.

UDP capabilities are demonstrated by broadcasting a sequence of packets onto the LAN, then listening for them and displaying their contents. The user may choose the number of packets to transmit.

Summary

Socket communications in Windows no longer needs to be painful. These wrapper classes can provide a solid foundation for many network communications applications, and will allow the programmer to concentrate on the communications instead of the myriad of details associated with normal Windows sockets programming.