

# WINSOCK 2

## WINDOWS SOCKET: PROGRAM EXAMPLES

### PART 1

Machine's OS is standalone Windows Xp Pro with SP2 except whenever mentioned. Compiler used was Visual C++ 2003 .Net 1.1. Beware the codes that span more than one line. Program examples have been tested for Non Destructive Test. All information compiled for Windows 2000 (NT5.0) above and...

1. Related functions, structures and macros used in the program examples have been dumped at [Winsock function & structure 1](#) and [Winsock function & structure 2](#).
2. Other related and required information (if any) not available in no. 2 can be found at [MSDN online](#).

#### Abilities

- Able to understand the basic of networking such as [TCP/IP](#).
- Able to understand Winsock implementation and operations through the APIs and program examples.
- Able to gather, understand and use the Winsock [functions](#), [structures](#) and [macros](#) in your programs.
- Able to build programs that use Microsoft C/Standard C programming language and Winsock APIs.

#### Windows Sockets 2 vs [Linux Sockets](#)

Windows Sockets version 2 (Winsock 2) used to create advanced Internet, intranet, and other network-capable applications to transmit application data across the wire, network protocol independent for Windows platforms. Winsock follows the Windows Open System Architecture (WOSA) model; it defines a standard service provider interface (SPI: WSP - Winsock Service Provider) between the application programming interface (API: WSA – Winsock API), with its exported functions and the protocol stacks. Winsock adapted for Windows starting from Windows Sockets 1.1 using the sockets paradigm that was first used and promoted by Berkeley Software Distribution (BSD) UNIX. That is why when you go through this documents you will find the similarity with Linux/UNIX Socket discussed in [Linux Socket](#). Winsock programming previously centered on [TCP/IP](#) but some programming practices that worked with TCP/IP do not work with every protocol. As a result, the Windows Sockets 2 API adds functions where necessary to handle several protocols. Windows Sockets 2 is designed to be used by C/C++ programmers (C++ used in MFC socket programming). Familiarity with Windows networking is required and for lengthy discussion of the TCP/IP, you can refer to [Advanced TCP/IP](#), Linux Sockets. Windows Sockets 2 can be used on all Windows platforms. In this section we will go through the

detail of the Winsock functions, structures and macros, and then followed by working program examples.

## Client and Server Communication

There are two distinct types of socket network applications: **Server** and **client**. In simple words, servers provide services for clients and other servers whereas clients request services from servers. Servers and clients have different behaviors; therefore, the process of creating them is different. What follows is the general model for creating a streaming TCP/IP server and client. The steps in creating (and difference between) server and client sockets are listed below.

Server side programs:

1. Initialize WSA – WSAStartup().
2. Create a socket – socket().
3. Bind the socket – bind().
4. Listen on the socket – listen().
5. Accept a connection – accept(), connect().
6. Send and receive data – recv(), send(), recvfrom(), sendto().
7. Disconnect – closesocket().

Client side programs:

1. Initialize WSA – WSAStartup().
2. Create a socket – socket().
3. Connect to the server – connect().
4. Send and receive data – recv(), send(), recvfrom(), sendto().
5. Disconnect – closesocket().

Some of the steps are similar for the server and client. These steps are implemented almost exactly alike. The steps in this guide will be specific to the type of application being created. Let explore some of the related functions, structures and macros available for us in Windows Sockets 2 and study the usage in the program examples.

Before you can compile and run the Windows Sockets programs you have to include the **ws2\_32.lib** library (or other libraries if needed) because the default installation of the Visual Studio / .Net does not add the library to the project by default. The following steps show how to add more libraries (dependencies) to your project. The first one is for Visual C++ .Net 1.1 and then followed by Visual C++ 6. Depend on the functions/structures/macros used in the programs and these similar steps can be used to add other non-default libraries to your project, else you may encounter a lot of errors during compilation/linking because the definition of the functions/structures/macros cannot be found and resolved.

For Visual Studio .Net (Visual C++ .Net 1.1): Select the **P**roject menu and the **your\_project\_name P**roperties... sub menu.

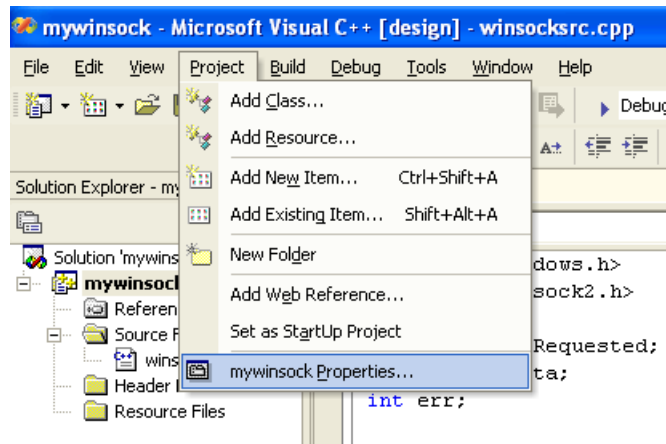


Figure 1

On the left window, expand the **Linker** folder and select the **Input** sub folder as shown below. For the **Additional Dependencies**, select the right empty field.

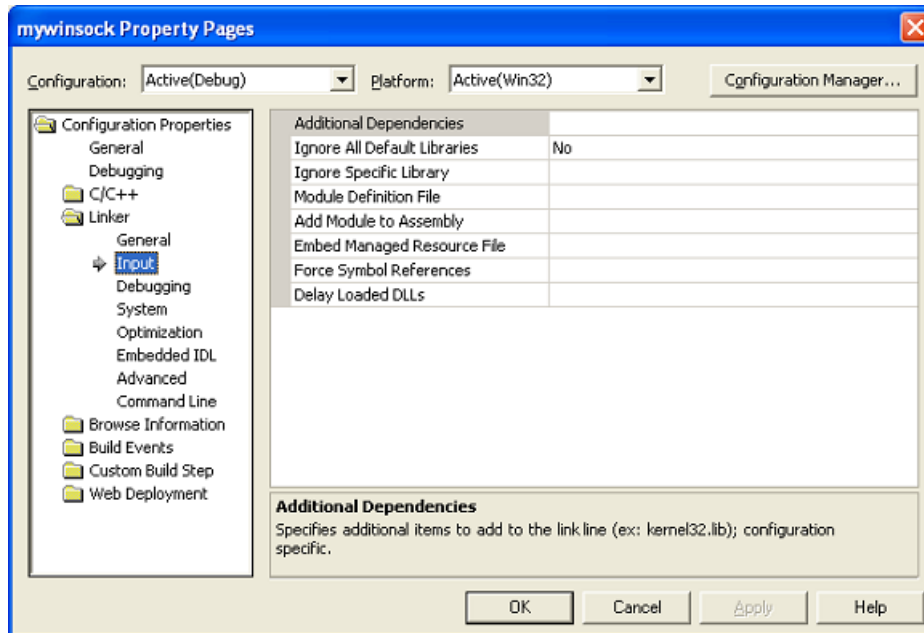


Figure 2

Type the library name, for this example, **ws2\_32.lib**. Then click the **OK** button twice to close all the project property pages.

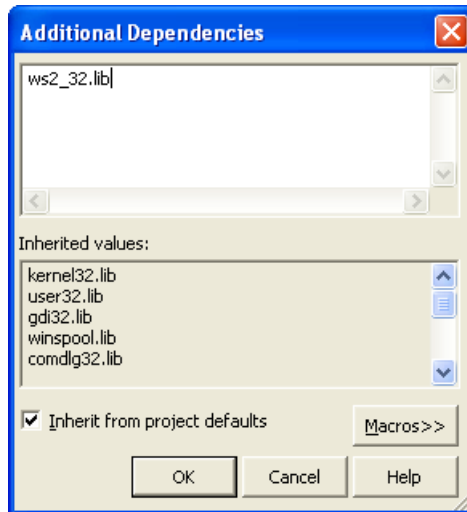


Figure 3

For Visual C++ 6.0, select the **Project** menu and then **Settings...** sub menu.

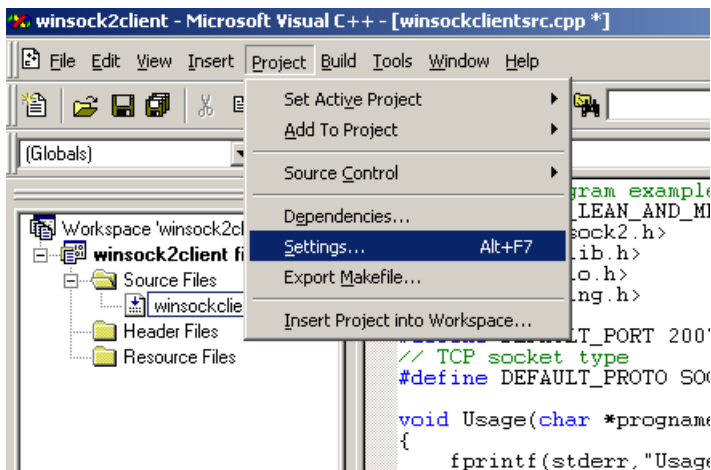


Figure 4

On the left window expand the root folder that is your project name and then select the **Link** tab. At the end of the **Object/library modules:** field, type the library name as shown below. Click the **OK** button to close the **Project Settings** page. Then you are ready to compile and run the program examples that follow. All examples compile and run successfully in **Debug** mode. If you run for **Release** mode, there are some errors related to the linking process but that is not our concern here.

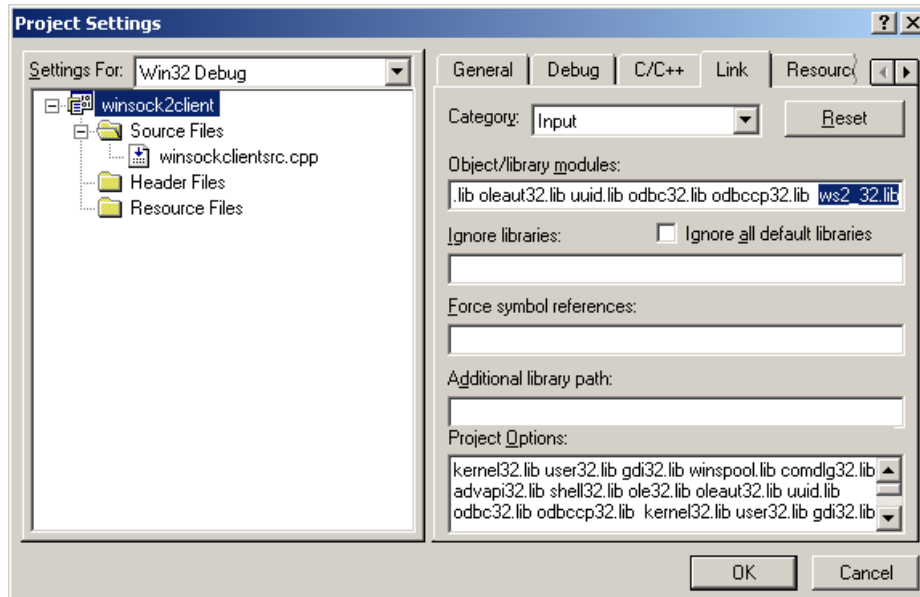


Figure 5

## Program Example

The following example demonstrates the use of the WSADATA structure and WSStartup() function. It shows how an application that supports only version 2.2 of Windows Sockets makes a WSStartup() call.

```
// Microsoft Development Environment 2003 - Version 7.1.3088
// Copyright (r) 1987-2002 Microsoft Corporation. All Right Reserved
// Microsoft .NET Framework 1.1 - Version 1.1.4322
// Copyright (r) 1998-2002 Microsoft Corporation. All Right Reserved
//
// Run on Windows XP Pro machine, version 2002, SP 2
//
// <windows.h> already included...
// WINVER = 0x0501 for Xp already defined in windows.h

#include <stdio.h>
#include <winsock2.h>

int main()
{
    WORD wVersionRequested;
    WSADATA wsaData;
    int wsaerr;

    // Using MAKEWORD macro, Winsock version request 2.2
    wVersionRequested = MAKEWORD(2, 2);

    wsaerr = WSStartup(wVersionRequested, &wsaData);
```

```

if (wsaerr != 0)
{
    /* Tell the user that we could not find a usable */
    /* WinSock DLL.*/
    printf("The Winsock dll not found!\n");
    return 0;
}
else
{
    printf("The Winsock dll found!\n");
    printf("The status: %s.\n", wsaData.szSystemStatus);
}

/* Confirm that the WinSock DLL supports 2.2.*/
/* Note that if the DLL supports versions greater */
/* than 2.2 in addition to 2.2, it will still return */
/* 2.2 in wVersion since that is the version we */
/* requested. */
if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2 )
{
    /* Tell the user that we could not find a usable */
    /* WinSock DLL.*/
    printf("The dll do not support the Winsock version %u.%u!\n",
LOBYTE(wsaData.wVersion),HIBYTE(wsaData.wVersion));
    WSACleanup();
    return 0;
}
else
{
    printf("The dll supports the Winsock version %u.%u!\n",
LOBYTE(wsaData.wVersion),HIBYTE(wsaData.wVersion));
    printf("The highest version this dll can support: %u.%u\n",
LOBYTE(wsaData.wHighVersion), HIBYTE(wsaData.wHighVersion));
}

/* The next task... */

return 0;
}

```

```

C:\> "f:\myproject\mywinsock\debug\mywinsock.e...
The Winsock dll found!
The status: Running.
The dll supports the Winsock version 2.2!
The highest version this dll can support: 2.2
Press any key to continue

```

Figure 6

Once an application or DLL has made a successful `WSAStartup()` call, it can proceed to make other Windows Sockets calls as needed. When it has finished using the services of the `WS2_32.DLL`, the application or DLL must call `WSACleanup()` to allow the `WS2_32.DLL` to free any resources for the application. Details of the actual Windows Sockets implementation are described in the `WSADATA` structure. An application or DLL can call `WSAStartup()` more than once if it needs to obtain the `WSADATA` structure information more than once. On each such call the application can specify any version number supported by the DLL.

An application must call one `WSACleanup()` call for every successful `WSAStartup()` call to allow third-party DLLs to make use of a `WS2_32.DLL` on behalf of an application. This means, for example, that if an application calls `WSAStartup()` three times, it must call `WSACleanup()` three times. The first two calls to `WSACleanup()` do nothing except decrement an internal counter; the final `WSACleanup()` call for the task does all necessary resource de-allocation for the task.