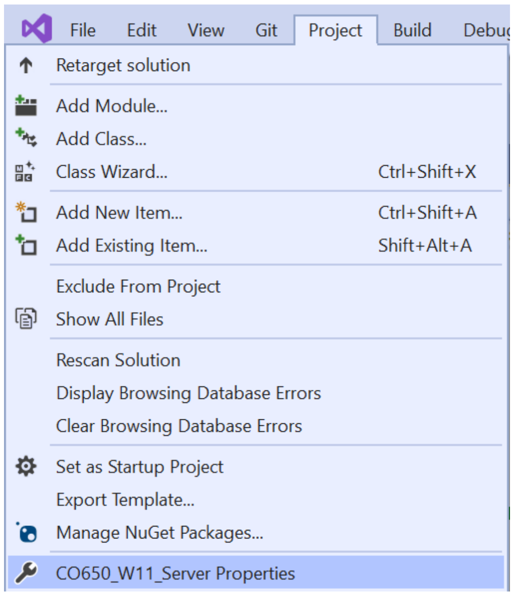
C++ Week 11 Sockets

Before you develop Windows Sockets programs within Visual Studio, you must include the **ws2_32.lib** library within your Project as the default installation of the Visual Studio / .Net does not include this API.

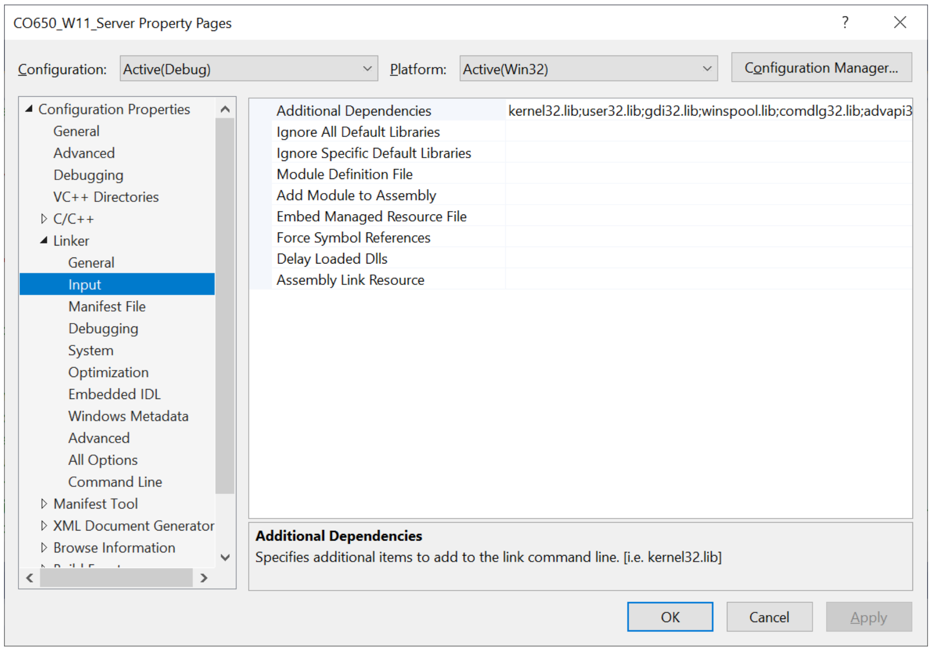
Create a new C++ Visual Studio Console Project and follow the steps below in order to add the Winsock library to your project.

Note: Before continuing ensure you save the project. If you don't the following menu options may not be available.

Select the **Project** menu and the **Properties…** sub menu



On the left window, expand **Configuration Properties -> Linker** and select the **Input** sub option as shown below. Select **Additional Dependencies** on the right and click on the arrow. Select the edit option.



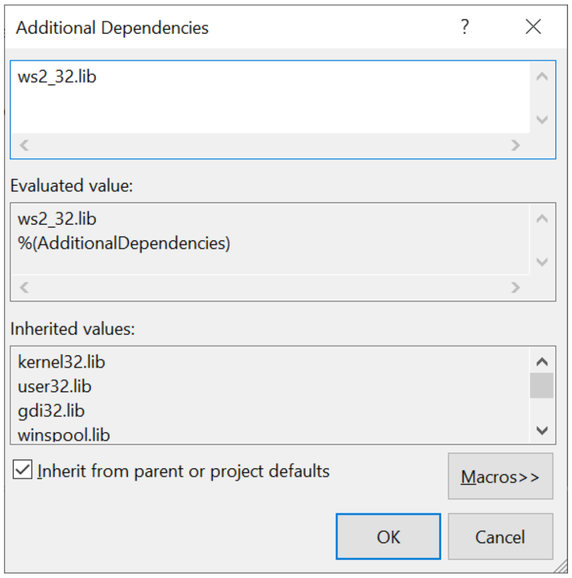
Configuration: Active(Debug)   Platform: Active(Win32)

Additional Dependencies	kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi3
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

Additional Dependencies
Specifies additional items to add to the link command line. [i.e. kernel32.lib]

C++ Week 11 Sockets

Type the library name, for this example, **ws2_32.lib**. Then click the **OK** button twice to close all the project property pages.



Additional Dependencies

ws2_32.lib

Evaluated value:

ws2_32.lib
%(AdditionalDependencies)

Inherited values:

kernel32.lib
user32.lib
gdi32.lib
winspool.lib

☑ Inherit from parent or project defaults

Server Code

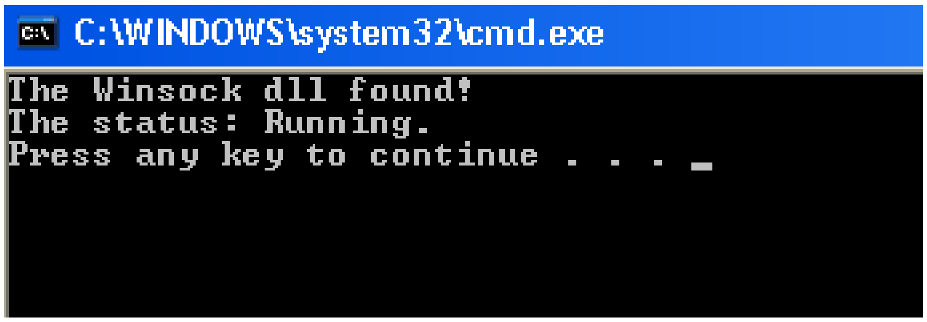
The following code demonstrates the use of the WSADATA structure and WSAStartup() function. It shows how an application that supports only version 2.2 of Windows Sockets makes a WSAStartup() call. Add the code below to your project and run the program to ensure that Winsock 2.2 is supported.

```
#include "stdafx.h"
#include <winsock2.h>
#include <ws2tcpip.h>
#include "iostream"

using namespace std;

int main(int argc, char* argv[]){
        SOCKET serverSocket, acceptSocket;
        int port = 55555;
        WSADATA wsaData;
        int wsaerr;
        WORD wVersionRequested = MAKEWORD(2, 2);
        wsaerr = WSAStartup(wVersionRequested, &wsaData);
        if (wsaerr != 0){
                cout << "The Winsock dll not found!" << endl;
                return 0;
        }
        else{
                cout << "The Winsock dll found!" << endl;
                cout << "The status: " << wsaData.szSystemStatus << endl;
        }
        return 0;
}
```

CC

C++ Week 11 Sockets

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. 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.
When you run the application you should get the output displayed below.

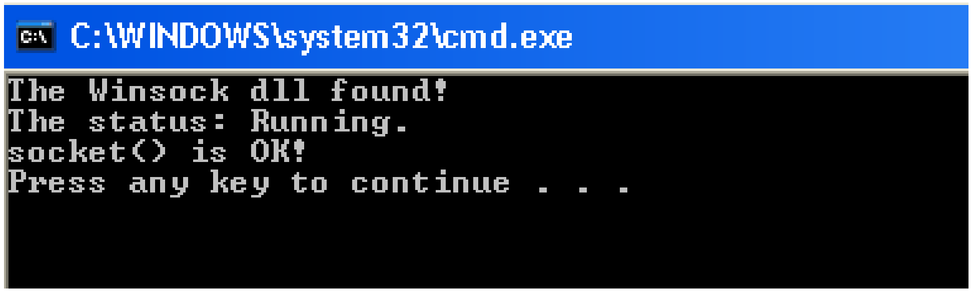


```
The Winsock dll found!
The status: Running.
Press any key to continue . . . _
```

Now add the code below (under the previous code but still within the main function). Here we create a new socket.

```
serverSocket = INVALID_SOCKET;
serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (serverSocket== INVALID_SOCKET){
        cout << "Error at socket(): " <<  WSAGetLastError() << endl;
        WSACleanup();
        return 0;
}
else {
        cout << "socket() is OK!" << endl;
}
```

Run the program once more and check the output is similar to that below.

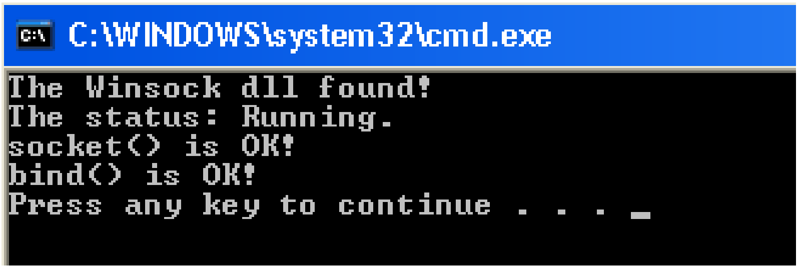


```
The Winsock dll found!
The status: Running.
socket() is OK!
Press any key to continue . . .
```

Now we must bind the socket to the servers IP address and a valid port.

```
sockaddr_in service;
service.sin_family = AF_INET;
InetPton(AF_INET, _T("127.0.0.1"), &service.sin_addr.s_addr);
service.sin_port = htons(port);
if (bind(serverSocket, (SOCKADDR*)&service, sizeof(service))== SOCKET_ERROR){
        cout << "bind() failed: " <<  WSAGetLastError() << endl;
        closesocket(serverSocket);
        WSACleanup();
        return 0;
}
else{
        cout << "bind() is OK!" << endl;
}
```

The output from your program should now resemble that below.

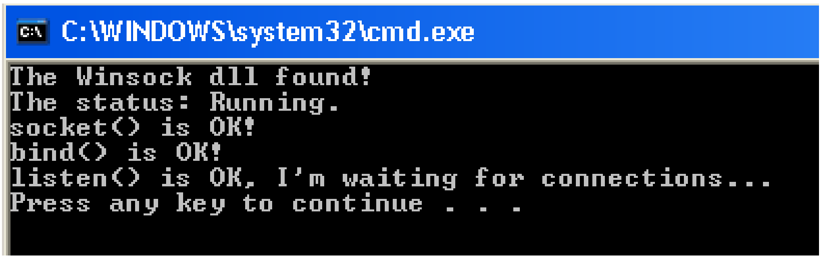


```
The Winsock dll found!
The status: Running.
socket() is OK!
bind() is OK!
Press any key to continue . . . _
```

Once the socket is bound to a port and IP address we can place it into a listening state.

```
if (listen(serverSocket, 1) == SOCKET_ERROR)
        cout << "listen(): Error listening on socket " <<  WSAGetLastError()<< endl;
else
        cout << "listen() is OK, I'm waiting for connections..." << endl;
```

Run the application once more.



C:\WINDOWS\system32\cmd.exe

```
The Winsock dll found!
The status: Running.
socket() is OK!
bind() is OK!
listen() is OK, I'm waiting for connections...
Press any key to continue . . .
```

The final step within the Server program is to append the following code to the main function. The accept function waits for client connection requests. On receiving one it returns a new socket that is connected to the client.

```
acceptSocket = accept(serverSocket, NULL,NULL);
if (acceptSocket == INVALID_SOCKET){
        cout <<  "accept failed: " << WSAGetLastError() << endl;
        WSACleanup();
        return -1;
}
cout << "Accepted connection" << endl;
system("pause");
WSACleanup();
```

If you run the program once more you should find the output within the console has not changed as the Server will only display "Accepted Connection" when a client has connected to it. We shall now create a client to test this.

Client Code

Now that we have created a basic server that accepts connections lets create a client application that will connect to the server. Start a new VS C++ Console Project. Add the ws2_32.lib library as before and include the same files within your source file as you did with the server.

Within main we invoke WSAStartup as we did within the server

```
SOCKET clientSocket;
int port = 55555;
WSADATA wsaData;
int wsaerr;
WORD wVersionRequested = MAKEWORD(2, 2);
wsaerr = WSAStartup(wVersionRequested, &wsaData);
if (wsaerr != 0){
	cout << "The Winsock dll not found!" << endl;
	return 0;
}
else{
	cout << "The Winsock dll found!" << endl;
	cout << "The status: " <<  wsaData.szSystemStatus << endl;
}
```

Create a socket

```
clientSocket = INVALID_SOCKET;
clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (clientSocket== INVALID_SOCKET){
	cout << "Error at socket(): " <<  WSAGetLastError() << endl;
	WSACleanup();
	return 0;
}
else {
	cout << "socket() is OK!" << endl;
}
```

Unlike the server there is no need to bind a client this is performed automatically by Winsock. We do however have to instruct the client to connect to the server.

```
sockaddr_in clientService;
clientService.sin_family = AF_INET;
InetPton(AF_INET, _T("127.0.0.1"), &clientService.sin_addr.s_addr);
clientService.sin_port = htons(port);
if (connect(clientSocket, (SOCKADDR*)&clientService, sizeof(clientService)) == SOCKET_ERROR){
        cout << "Client: connect() - Failed to connect." << endl;
        WSACleanup();
        return 0;
}
else   {
       cout << "Client: connect() is OK." << endl;
       cout << "Client: Can start sending and receiving data..." << endl;
}
system("pause");
WSACleanup();
return 0;
```

Now run the server first and then the client program, you should see confirmation on the server and client console windows that a connection has been established.

Server Code

It may be useful for the server to know details about the client that has been connected.

Add the following declarations to the top of the server's main function.

```
SOCKADDR_STORAGE from;
int retval,fromlen,socket_type;
char servstr[NI_MAXSERV], hoststr[NI_MAXHOST];
```

Add the following three lines just above the accept function invocation and change the arguments passed to accept in line with that below.

C++ Week 11 Sockets

Note:: Don't leave two accept calls within your code, you should only have one.

```
fromlen = sizeof(socket_type);
retval = getsockopt(serverSocket, SOL_SOCKET, SO_TYPE, (char *)&socket_type, &fromlen);
fromlen = sizeof(from);
acceptSocket = accept(serverSocket, (SOCKADDR *)&from, &fromlen);
```

Add the following statements directly above the statement that prints "Accepted Connection" and run both applications (server first).

```
retval = getnameinfo((SOCKADDR *)&from,
                     fromlen,
                     hoststr,
                     NI_MAXHOST,
                     servstr,
                     NI_MAXSERV,
                     NI_NUMERICHOST | NI_NUMERICSERV);
if (retval != 0){
            cout <<  "getnameinfo failed: " << retval << endl;
                WSACleanup();
            return -1;
}

cout << "Accepted connection from host " <<  hoststr << " and port " << servstr << endl;
```

Once more run the server and then the client. The clients details should now be displayed on the server's console window.