Compiling a C Windows Sockets Program in Microsoft Visual Studio Community 2019

Last Update: 2/5/20

These instructions pertain to Microsoft Visual Studio Community 2019. The process that you follow would be similar with Visual Studio 2019 Professional and Enterprise Editions, and with older versions of Visual Studio. Note that installing Visual Studio using a slow Internet connection can take up to an hour.

The C programming language is the predecessor to C++, and is in many ways a "subset" of C++. C code can be compiled with a C++ compiler. There are some differences between the two languages, most notably the lack of support for objects in C. C is widely used as a "low-level" language for system programming, embedded systems, drivers, and other applications that need to run fast with a small memory footprint. With just a bit of tweaking, the Microsoft Visual Studio Community 2019 or the full Visual Studio 2019 IDE can compile C source code, link to the appropriate C libraries and generate .exe files. This document explains how to do that for a program that uses Windows Sockets. If you are proficient with C++ in Visual Studio, feel free to deviate from the following instructions to accommodate your existing Visual Studio configuration. If you do that, you will still need to make the changes shown on pages 2, 3 and 5.

Uninstall any old versions of Visual Studio. Download *Microsoft Visual Studio Community 2019* from https://visualstudio.microsoft.com/downloads/. Run the installer and click through the preliminary defaults. When *Installing – Visual Studio Community 2019…* displays, select **Individual components** near the top. Scroll down to **Compilers, build tools, and runtimes**, *carefully* check *C++/CLI support for v142 build tools (14.21)*, and click **Install** at the lower right. When the *Do you want to continue without workloads?* dialog box opens, click **Continue**. You may also wish to install other Visual Studio components at this time. (If you already have Visual Studio 2019 installed, verify that C++/CLI support is installed, and add this component if it is not present.)

Visual Studio will start up after installation. You can sign in if you have a Microsoft account. If not, click "Not now, maybe later" at the bottom of the Visual Studio Welcome pane. The initial license will expire in 30 days. If you plan to continue using Visual Studio after that, click the account settings button at the upper right corner of the main IDE screen to enter you Microsoft account information and remove the 30 day restriction.

Creating a Project

Start Visual Studio, sign in if necessary using your Microsoft account, and choose **Create a new project** at the lower right**.**

Under *All Languages*, select **C++**. Under *All Platforms*, select **Windows**. Under All *Project Types*, choose **Console**. If this is your first C/C++ project, you will see a "Not finding what you're looking for?" message to the lower right. If so, click on "Install more tools and features", and wait for any updates to install.

When *Modifying – Visual Studio Community 2019…* displays, click on *Desktop development with C++* below. Take the defaults at the right, then click **Modify** and then **Continue**. After the installation is finished, reboot.

Restart Visual Studio, select **Create a New Project**, **Console App** and **Next**. Rename your project as desired and click **Create**. If you don't want to create your project in the default folder, browse to the location where you want to store the project files or enter a new folder name. Optionally check *Place solution and project in the same directory*. Then click **Create**.

Adding the Windows Sockets Library to the Project

`ws2_32.lib` contains the Windows Sockets functions used by current versions of the Windows operating system. To access those functions, you must tell Visual Studio to reference `ws2_32.lib` at link time. This must be done each time you create a new Windows Sockets project. If you build separate Debug and Release versions of a program, you will need to add this library to each version of the project.

Right click on your project name in the Solution Explorer and click **Properties | ►Linker | Input**. Find *Additional Dependencies* at the right, and click on the list of library files to the right of that (`kernel32.lib`, etc.). Click the arrow that appears to the right of the list of library files, and select *<Edit…>*, as shown in Figure 1. In the *Additional Dependencies* window, enter `ws2_32.lib`, as shown in Figure 2, and click **OK**. Verify that `ws2_32.lib` has been added to the front of the *Additional Dependencies* list, as shown in Figure 3.

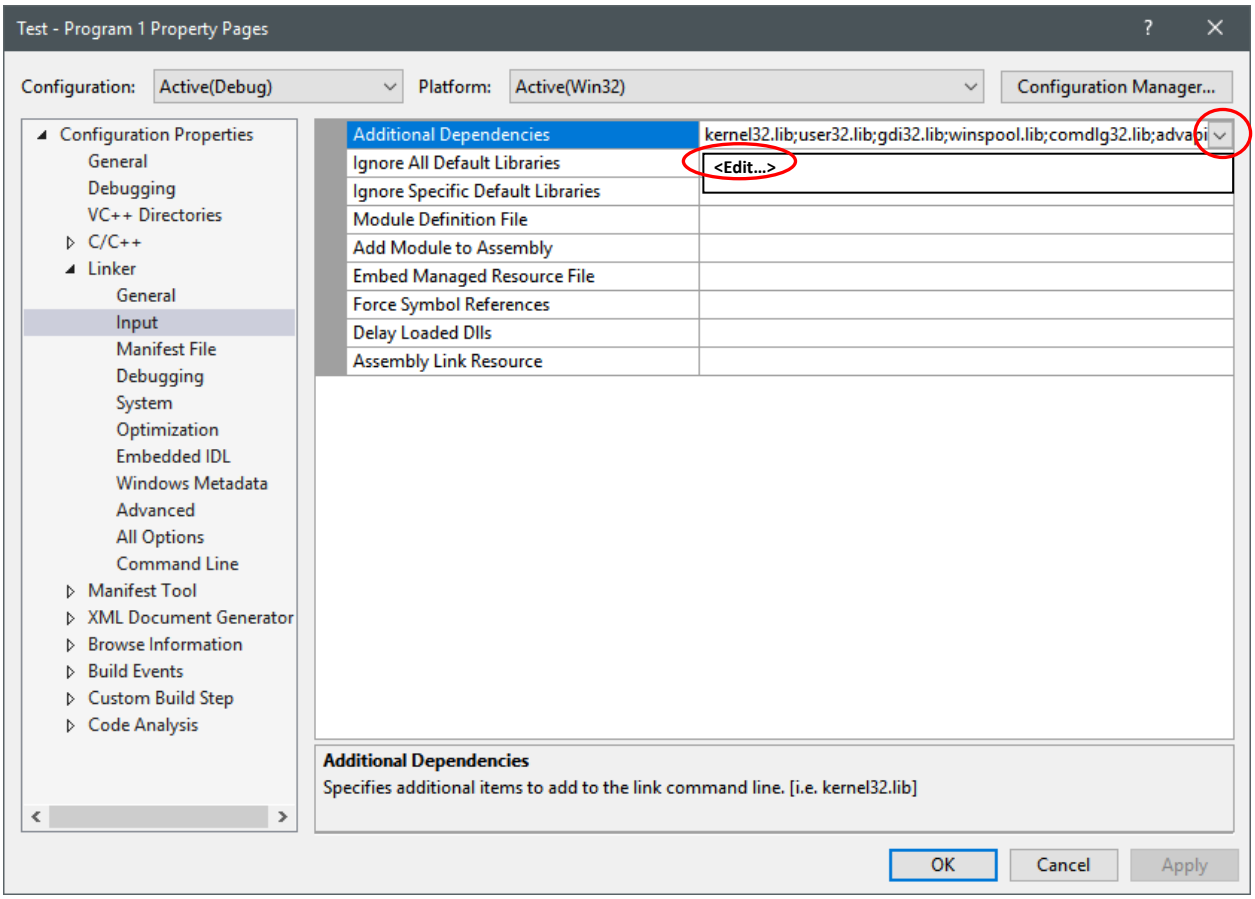


Figure 1. Adding a Library

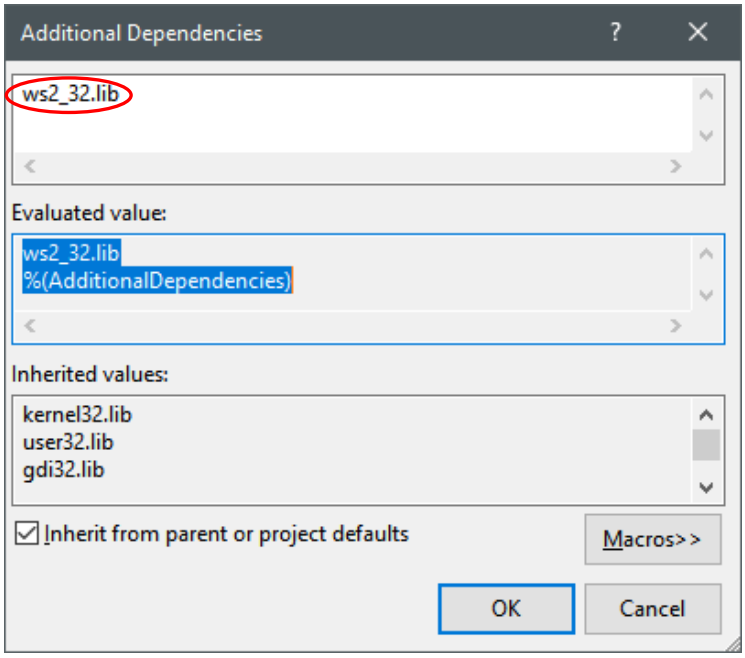


Figure 2. The Additional Dependencies Window

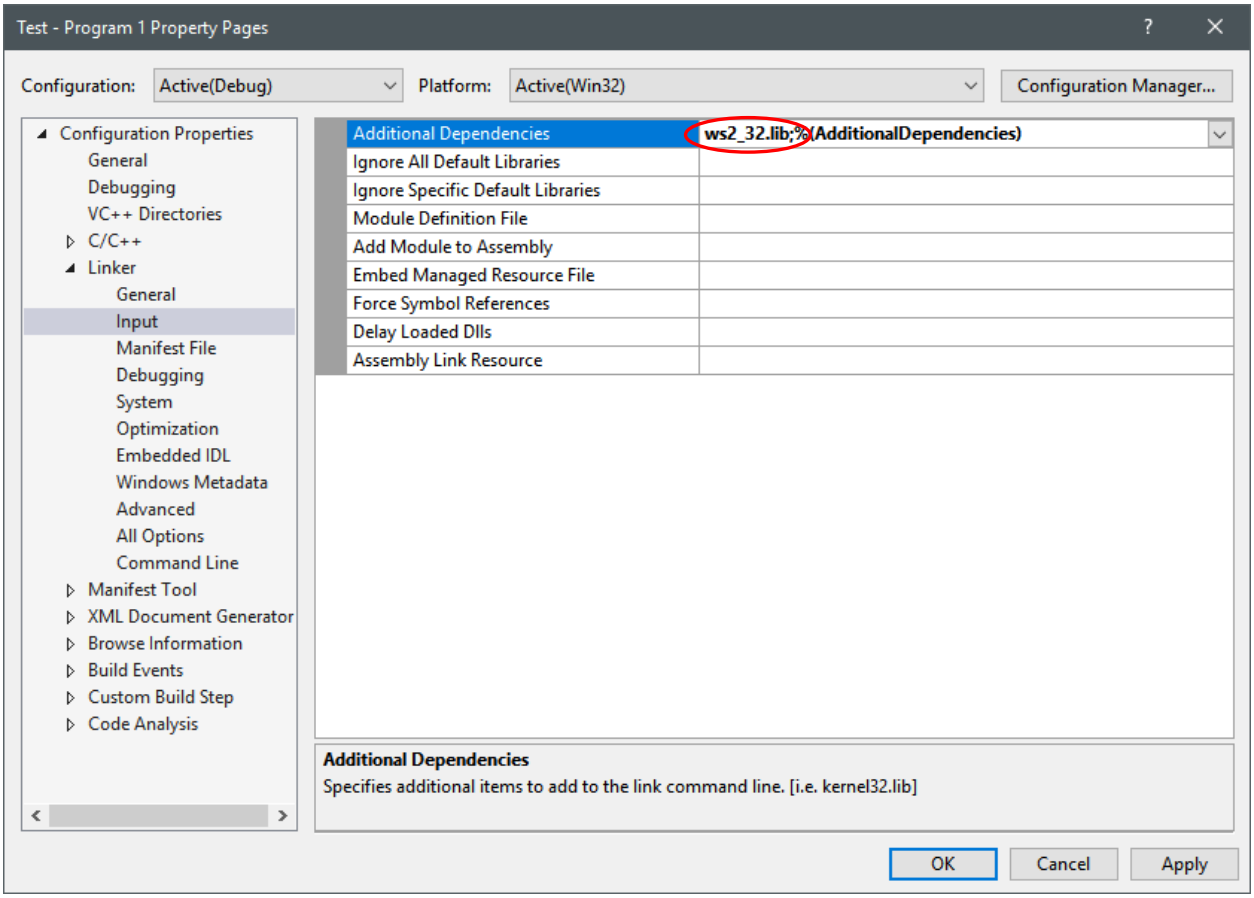


Figure 3. `ws2_32.lib` Has Been Added

Adding Source Code Files to the Project

Under *Source Files* in the Solution Explorer, right click on any preexisting `*.cpp` and `*.h` files and remove them.

Add a new (empty) source code file to the project by right clicking on *Source Files* in the Solution Explorer and choosing **Add | New Item…** Expand *Visual C++* under *Installed*, select *Code*, then select *C++ File (.cpp)*, and enter your source code file name below as `<filename>.c` (not `*.cpp`). For example, `WSEchoClientv6.c`. Click **Add** and verify that the filename appears under *Source Files* in the Solution Explorer. Type your code into the Editor window at the left.

To add an existing `*.c` file to your project, right click on *Source Files* in the Solution Explorer, then click **Add | Existing Item…** Browse to the existing C source file, select it and click **Add**. (Note that a file added this way to an existing project is not copied, just referenced. If you edit a source code file added this way from within Visual Studio, the original file will be modified.)

Save your project files frequently using **File | Save All**.

Header Files

To reference the C header (`*.h`) file associated with the `ws2_32.lib` Windows Sockets library, add the following preprocessor directives near the beginning of your program, before the `main()` function:

```
#include <winsock2.h>
#include <ws2tcpip.h>
```

These `#include` directives must be included in any source code (`*.c`) file that uses Windows Sockets functions.

Compiling and Linking Your Program

Building your project in Visual Studio involves compiling your source code files and linking them to any referenced libraries. To compile a version suitable for testing in the Visual Studio debugger, click **Build | Build Solution**, just as you would with a C++ program. The output from the build, along with any compiler or linker error messages, will be displayed in the *Output* window below the Editor. After a successful build, you should see this message:

```
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

(CONTINUED ON NEXT PAGE)

Executing Your Program

If you run your program in the Visual Studio debugger you will need to provide command line parameters to the debugger as follows. In Solution Explorer, right click on the project name, then click **Properties | Configuration Properties | Debugging**. Add the arguments in the box to the right of *Command Arguments* (as shown in Figure 4), using the same syntax as you would on the command line (including double quotes around text string arguments), and click **OK**. *Do not include the program executable name, which the command line processor parses as* `arg[0]`. To run your program from the debugger, press F5.

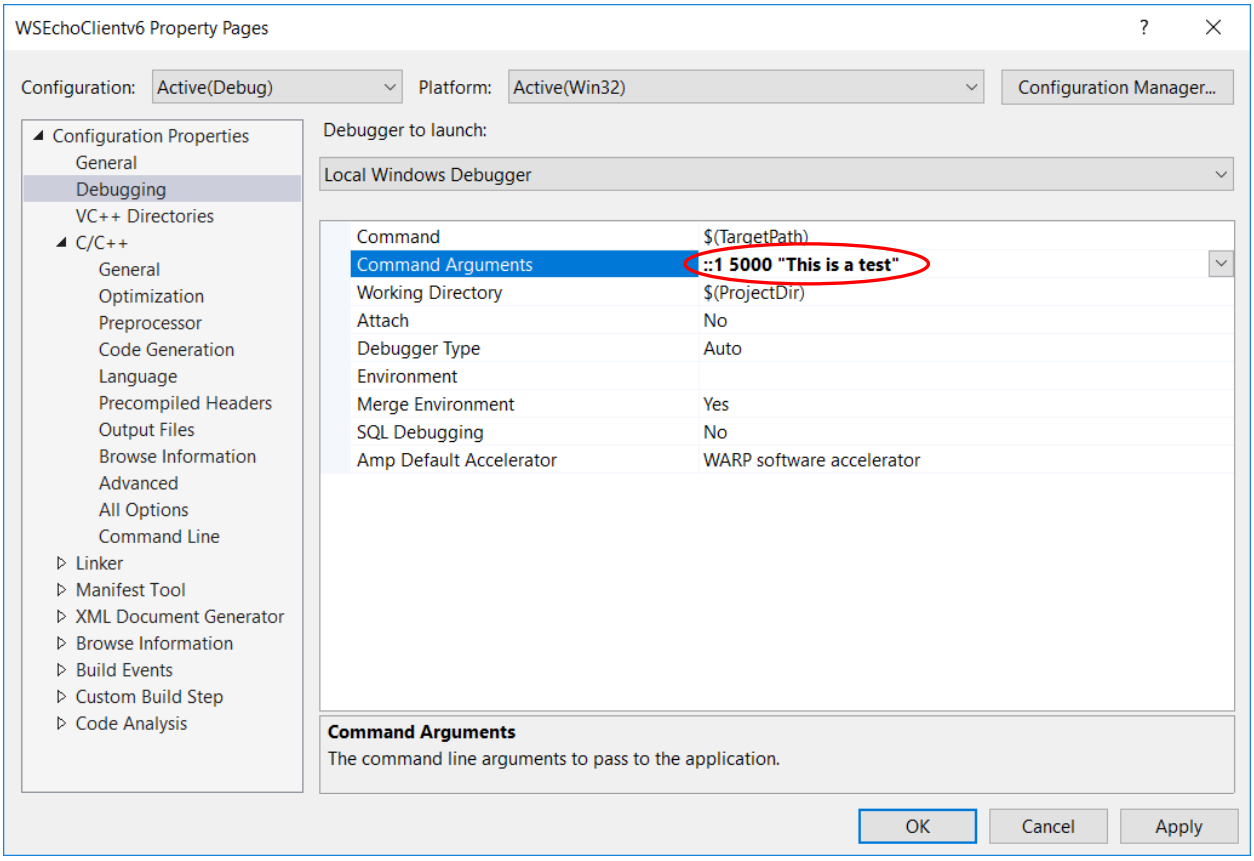


Figure 4. Adding Command Line Parameters When Using the Debugger

Testing a client/server application typically involves running separate client and server programs, only one of which can be executed in the debugger at a given time. The other will need to be executed from the Windows command line. (When command line arguments must be entered, starting a program from File Explorer doesn't work very well.)

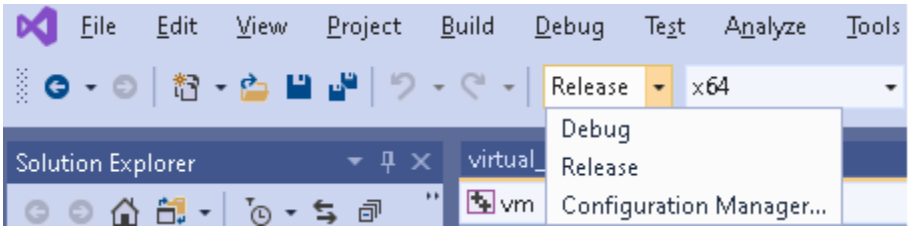
You may find it more convenient to run both the client and server processes from the Windows command line. If so, enter the arguments after the program name, separated by spaces. You'll find the executable file containing your program in a subfolder under your project folder with a default path similar to `C:\Users\<username>\source\repos\<project name>\Debug\`

To track program execution and debug runtime errors without using the Visual Studio debugger, try adding `printf()` and `getchar()` statements similar to the following:

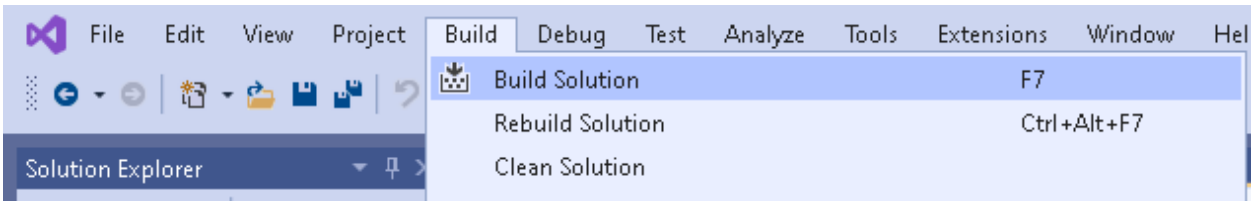
```
printf("Connection established with server.  Press any key to continue.");
getchar();
```

Compiling a Release Version

You may wish to compile a debug version of the programs for testing purposes, but the final version that you submit must be a release version. To compile a release version, change Configuration type to **Release** in the Dropdown menu as shown.



Then Build the Project



You'll find the Release mode executable in a subfolder under your project folder with a default path similar to

X86: `C:\Users\<username>\source\repos\<project name>\Release\` or
X64: `C:\Users\<username>\source\repos\<project name>\x64\Release\`

Firewall Issues

The ephemeral port number that you choose to use for your server may be blocked by the firewall on your computer. When you run your program for the first time, the firewall will usually request permission to unblock the port, which you can safely allow. Occasionally a "silent" firewall will block the port without alerting you. In that case, you may have to use the firewall management interface in combination with the Visual Studio debugger and/or Wireshark to locate the problem.

Summary

Here's a quick summary of the main steps. Refer to the previous sections for details:

1. Create the project as a Win32 Console Application.
2. Add `ws2_32.lib` to the project dependencies.
3. Delete any preexisting `*.cpp` files in your project.
4. Add new or existing source code (`*.c`) files to the project.
5. Add command line parameters to the debugger configuration (if desired).
6. Build the project.
7. Test from the command line or in the debugger.