Using libcurl with SSH support in Visual Studio 2008

Version 1.1

Revision History

Revision	Date	By	Comment
1.0	08 Dec. 2008	Andrei Jakab	Initial version

1.1	07 Jan. 2008	Andrei Jakab	• section 3.4, item 3.b.ii:
			  ○ replaced CURL_STATIC by CURL_STATICLIB
			  ○ added CURL_DISABLE_LDAP
			• section 4.3.2, item 4.b:
			  ○ removed *wldap32.lib*

Table of Contents

Conventions

The following font conventions are used in this document:

- *italic* is used for filenames, directory names, and URLs
- `constant width` is used to indicate commands and code sections
- **red, bold and underlined text** indicates important items
- **bold** is used to represent GUI items (e.g. menus, menu items, list nodes etc.)

1. Introduction

Libcurl is a widely-used open-source library for transferring files. It supports many protocols (e.g. FTP, HTTP, SFTP etc.) and it is very well designed.

One of libcurl's strengths is its portability. You can build it on numerous platforms and you can be sure that it will work the same way on all of them. This wide support also means that the developers cannot constantly update the readme files for all the supported platforms. Thus, I have decided to create this document in order to share my experiences while compiling a static version of the curl library with SSH support in Visual Studio 2008.

This guide is inspired from Rosso Salmanzadeh's excellent "Using libcurl in Visual Studio" guide.

2. Downloading the latest software

Libcurl requires two additional open-source libraries in order to provide the SSH functionality: libSSH2 and OpenSSL. Also, since the compilation of OpenSSL makes use of Perl scripts, you need to have a Perl distribution installed on your machine. I have chosen ActivePerl, a free distribution by ActiveState, for this purpose.

The OpenSSL and libSSH2 libraries are distributed as tarballs compressed using gzip. Hence, you will need a utility to uncompress and subsequently open the tar file. I recommend the open-source archiver 7-Zip.

Software	URL	Current Version
ActivePerl	http://www.activestate.com/downloads/index.mhtml	5.10.0.1004
OpenSSL	http://www.openssl.org/source	0.9.8i
libSSH2	http://www.libssh2.org	0.18
libcurl	http://curl.haxx.se/download.html	7.19.2
7-Zip	http://www.7-zip.org	4.62

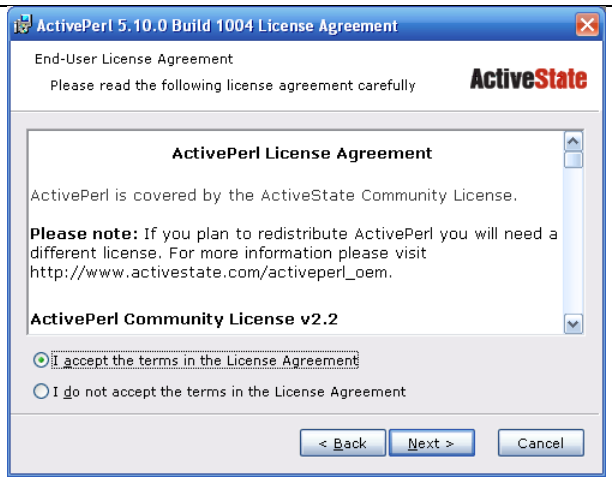
NOTE: This guide assumes that you already have successfully installed Microsoft Visual Studio 2008 (VS2008) and the Windows SDK. The examples in this document are built using version 9.0.30729.1 SP of VS2008 and Version 6.1 of the Windows SDK.

3. Installation / Compilation

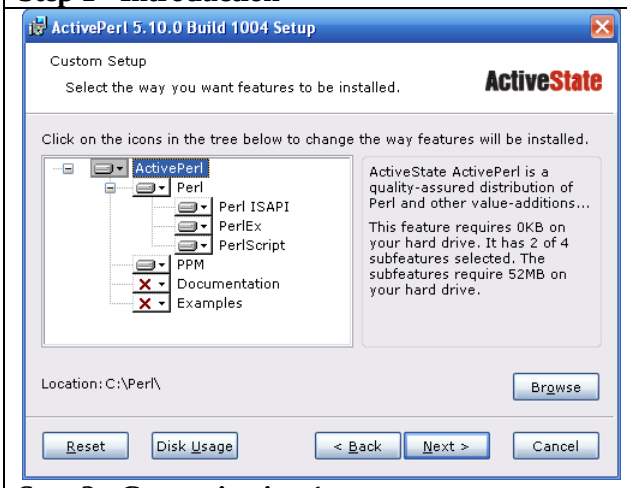
3.1 ActivePerl

Installing ActivePerl is fairly straightforward. Below are screenshots from each step of the installation process. Please pay particular attention to Step 4 because choosing the wrong option there will make your life harder later.

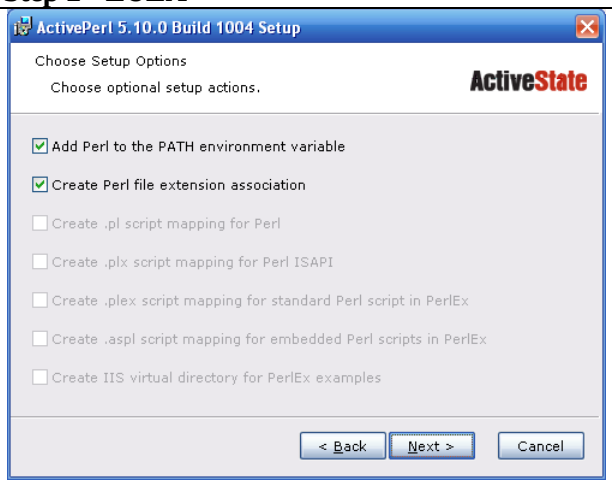
Step 1 - Introduction



Step 2 - EULA



Step 3 - Customization 1

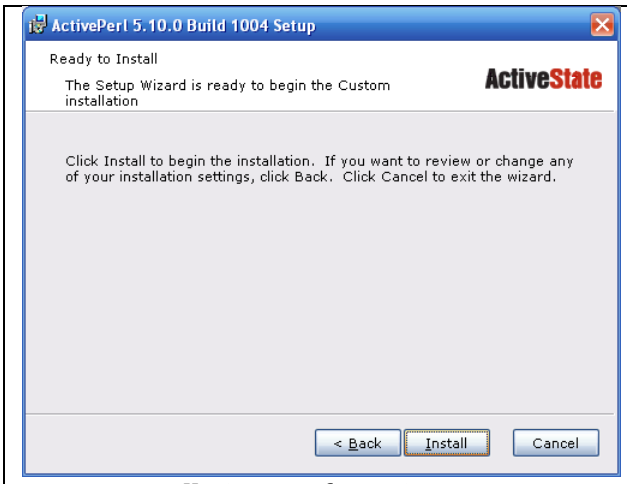


Step 4 - Customization 2

Note 1: The "Documentation" and "Examples" are only useful if you wish to learn how to use ActivePerl; for the purposes of this document, these items are not needed.

Note 2: The default installation location is not in the Program Files" Some clean freaks (such as yours truly) might take offence to this.

NOTE: Make sure that there's a checkmark besides the "Add Perl to the PATH environment variable" option.



Step 5 - Installation Confirmation

Step 6 - Installation Complete

3.2 OpenSSL

Compiling the OpenSSL library is a bit tricky. The following step-by-step guide should help you get through the compilation process as quickly and as painlessly as possible.

1. Extract the *openssl-0.9.8i.tar* file from the gzip file to a temporary directory
2. Extract the *openssl-0.9.8i* folder from the tar file and place it in the C: root directory
3. Open a Visual Studio 2008 command prompt (if you've installed VS 2008 with the default settings, the command prompt shortcut should be located in **Start** -> **All Programs** -> **Microsoft Visual Studio 2008** -> **Visual Studio Tools** -> **Visual Studio 2008 Command Prompt**)
4. Create the directory where the output of the compilation process will be stored:
   `mkdir c:\openssl_lib`
5. Change the working directory to the OpenSSL directory:
   `cd c:\openssl-0.9.8i`
6. Configure the OpenSSL installation with:
   `perl Configure VC-WIN32 --prefix=c:/openssl_lib`
   where the `--prefix` argument specifies where OpenSSL header and library files will be copied at the end of the compilation process.

   NOTE: The path that is passed to the `--prefix` argument must be in the UNIX format i.e. forward slashes are used to separate directories and not backward slashes like it is customary in Windows.

The output of this command should look like this:

7. Create the required assembly files:
   `ms\do_masm`
   Note: Using assembly files makes the execution of library functions much faster. If you do not wish to use assembly files, use the following command instead and jump to step 8:
   `ms\do_ms`
8. Compile the static library:
   `nmake -f ms\nt.mak`
9. The compilation process takes a while so you can go grab some coffee at this point.
10. If all is well, at the end of the compilation you will have some libraries and a number of executables in *C:\openssl-0.9.8i\out32*
11. The library contains some built in tests that allow you to check if everything has compiled properly and if the library is in working order:
    `nmake -f ms\nt.mak test`
    If the library has compiled properly, you should obtain a "passed all tests" message once the command finishes executing.
12. To install OpenSSL to the location you specified in step 6, run:
    `nmake -f ms\nt.mak install`

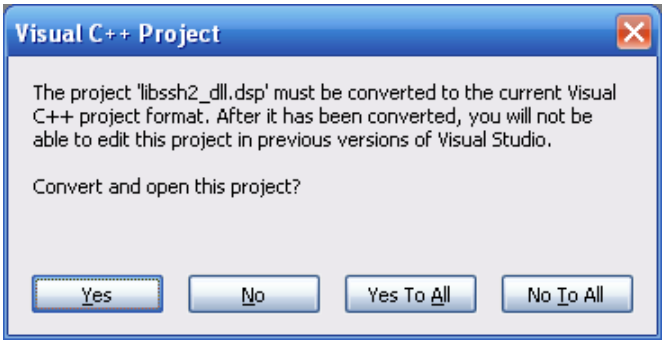
13. Before moving on to the next library, there one line of code needs to be added to one of the OpenSSL header files. Open the file *C:\openssl_lib\include\openssl\ossl_typ.h* in the Visual Studio 2008 IDE and add the following after line 102:
    `#undef OCSP_RESPONSE`
    After this insert, lines 98 - 103 should look like this:

```
98  #ifdef OPENSSL_SYS_WIN32
99  #undef X509_NAME
100 #undef X509_EXTENSIONS
101 #undef X509_CERT_PAIR
102 #undef PKCS7_ISSUER_AND_SERIAL
103 #undef OCSP_RESPONSE
104 #endif
```

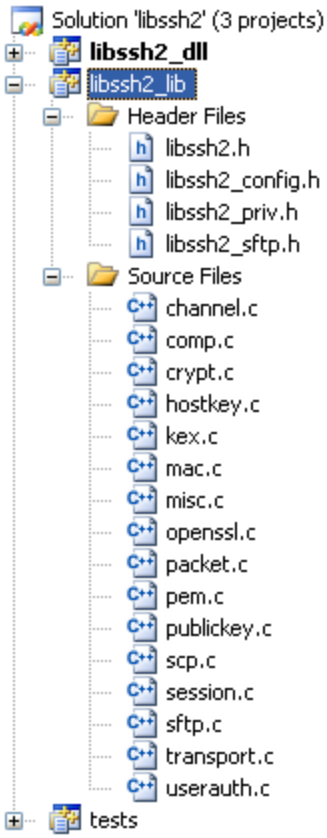
Note: You can find additional compiling instructions in *C:\openssl-0.9.8i\INSTALL.W32*; this file also contains a troubleshooting section that could help you out if something goes wrong during the compilation process.

3.3 libSSH2

1. Extract the *libssh2-0.18.tar* file from the gzip file to a temporary directory.
2. Extract the *libssh2-0.18* directory from the tar file and place it in the C: root directory.
3. Open *C:\libssh2-0.18\win32\libssh2.dsw* in the Visual Studio 2008 IDE. Since this is a Visual Studio 6 workspace, the following message box will appear:



Click on "Yes To All". Once the conversion is completed, you should see the following in the **Solution Explorer** window:



4. Open libssh2_config.h and replace line 46 with:

```
#define ssize_t SSIZE_T
#define uint32_t UINT32
```

Also, unless you wish to use the zlib compression library, comment out line 60.

After these edits, lines 44 - 60 should look like this:

```
44  #ifdef _MSC_VER
45  #define snprintf _snprintf
46  #define ssize_t SSIZE_T
47  #define uint32_t UINT32
48  #define strncasecmp _strnicmp
49  #define strcasecmp _stricmp
50  #else
51  #ifdef __MINGW32__
52  #define WINSOCK_VERSION MAKEWORD(2,0)
53  #else
54  #define strncasecmp strnicmp
55  #define strcasecmp stricmp
56  #endif /* __MINGW32__ */
57  #endif /* _MSC_VER */
58
59  /* Compile in zlib support */
60  //#define LIBSSH2_HAVE_ZLIB 1
```

5. Open libssh2.h and replace line 54 with:

```
#    define LIBSSH2_API// __declspec(dllexport)
```

After this edit, lines 50 - 61 should look like this:

```
50  /* Allow alternate API prefix from CFLAGS or calling app */
51  #ifndef LIBSSH2_API
52  # ifdef LIBSSH2_WIN32
53  #  ifdef LIBSSH2_LIBRARY
54  #   define LIBSSH2_API// __declspec(dllexport)
55  #  else
56  #   define LIBSSH2_API __declspec(dllimport)
57  #  endif /* LIBSSH2_LIBRARY */
58  # else /* !LIBSSH2_WIN32 */
59  #  define LIBSSH2_API
60  # endif /* LIBSSH2_WIN32 */
61  #endif /* LIBSSH2_API */
```

6. Next, you must tell the compiler and the librarian where the OpenSSL library is located.
    a. Right-click on the libssh2_lib project in the **Solution Explorer** window and select **Properties** from the pop-up menu.
    b. Expand the **Configuration Properties** node, then the **C/C++** node, and select **General**. Choose the **Additional Include Directories** property and add the following:
    ,*C:\openssl_lib\include*
    (the comma is needed since there are other paths already present)
    c. In the **Configuration Properties** node, expand the **Librarian** node, and select **General**.
        i. Change the **Output File** property to:
        Debug_lib\libssh2.lib
        ii. Click on the **Additional Dependencies** property and set it to:
        libeay32.lib ssleay32.lib
        iii. Choose the **Additional Library Directories** and add the following path:
        C:\openssl_lib\lib
    d. Click on the **OK** button.
7. Now we are ready to compile. Right-click on the libssh2_lib project and select **Build** from the pop-up menu.
8. The compiler will display a couple of security warnings since the authors of the libSSH2 library did not use the security-enhanced versions of the Visual Studio CRT functions. There will also be some “possible loss of data” warnings. For our purposes, both of these types of warnings can be safely ignored.

3.4 libcurl

1. Extract the *curl-7.19.2* folder from the archive and place it in the C: root directory. For our purposes, we will need the files located in the *lib* and *include* directories.

2. Open *C:\curl-7.19.2\lib\curllib.vcproj* in the Visual Studio 2008 IDE. Since this is a Visual Studio 2005 project file, the "Visual Studio Conversion Wizard" will appear. The wizard should not encounter any problems converting the project to the VS2008 format. You will only get a warning concerning the User Account Control (UAC) feature of Windows Vista. Since we are building a library and not an executable, we can safely ignore this warning.

3. Next, you must tell the compiler and the librarian to use the libSSH2 library and where this library is located:
    a. Right-click on the curllib project in the **Solution Explorer** window and select **Properties** from the pop-up menu.
    b. Expand the **Configuration Properties** node, then the **C/C++** node.
        i. Click on the **General** node. Next to **Additional Include Directories**, type in the following:
        ,C:\libssh2-0.18\include
        (the comma is needed since there is already one additional include directory specified)
        ii. Select the **Preprocessor** node and click on the **Preprocessor Definitions** property. Add in the following to the existing definitions:
        ;CURL_STATICLIB;USE_LIBSSH2;CURL_DISABLE_LDAP;HAVE_LIBSSH2;HAVE_LIBSSH2_H;LIBSSH2_WIN32 ;LIBSSH2_LIBRARY
    c. In the **Configuration Properties** node, expand the **Librarian** node, and select **General**.
        i. Select **Additional Dependencies** and type in: *libssh2.lib*
        ii. Next to **Additional Library Directories**, type in the path: *C:\libssh2-0.18\win32\Debug_lib*
    d. Click on the **OK** button.
4. After all this work, we are finally ready to compile the libcurl library. Right-click on the curllib project in the **Solution Explorer** and select **Build** from the pop-up menu. VS2008 will prompt you to save the solution file that was created for this project. Once you save it, compilation will begin:

    1>------ Build started: Project: curllib, Configuration: Debug Win32 ------
    1>Compiling...
    1>connect.c
    1>content_encoding.c
    1>cookie.c
    ...

```
1>version.c
1>base64.c
1>Generating Code...
1>Creating library...
1>Creating browse information file...
1>Microsoft Browse Information Maintenance Utility Version 9.00.21022
1>Copyright (C) Microsoft Corporation. All rights reserved.
1>Build log was saved at "file://c:\curl-7.19.2\lib\Debug\BuildLog.htm"
1>curllib - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

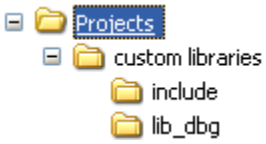
By default, the output directory is *C:\curl-7.19.2\lib\Debug*. In order to create an application that uses the libcurl library, we only need the *curllib.lib* file from the output directory and the *C:\curl-7.19.2\include\curl* directory, which contains the library's header files.

4. Using the libcurl library in your Visual Studio project

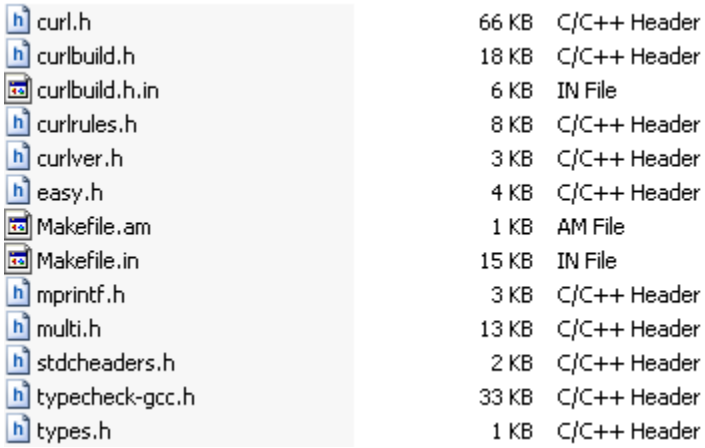
In this section we will create a test project, which will at the same time test the library's functionality and demonstrate how to integrate libcurl into one of your projects. I suggest that you follow the example step by step (i.e. use the same project names, paths etc.) so that you obtain the same figures as the ones shown below. This will make your life easier in case you encounter any problems along the way.

4.1 Preparing the project's file structure

Create the following folder structure in you C: root directory:



Copy the *curl* folder from the *C:\curl-7.19.2\include* directory into the *C:\Projects\custom libraries\include* directory. The *curl* directory should contain the following files:



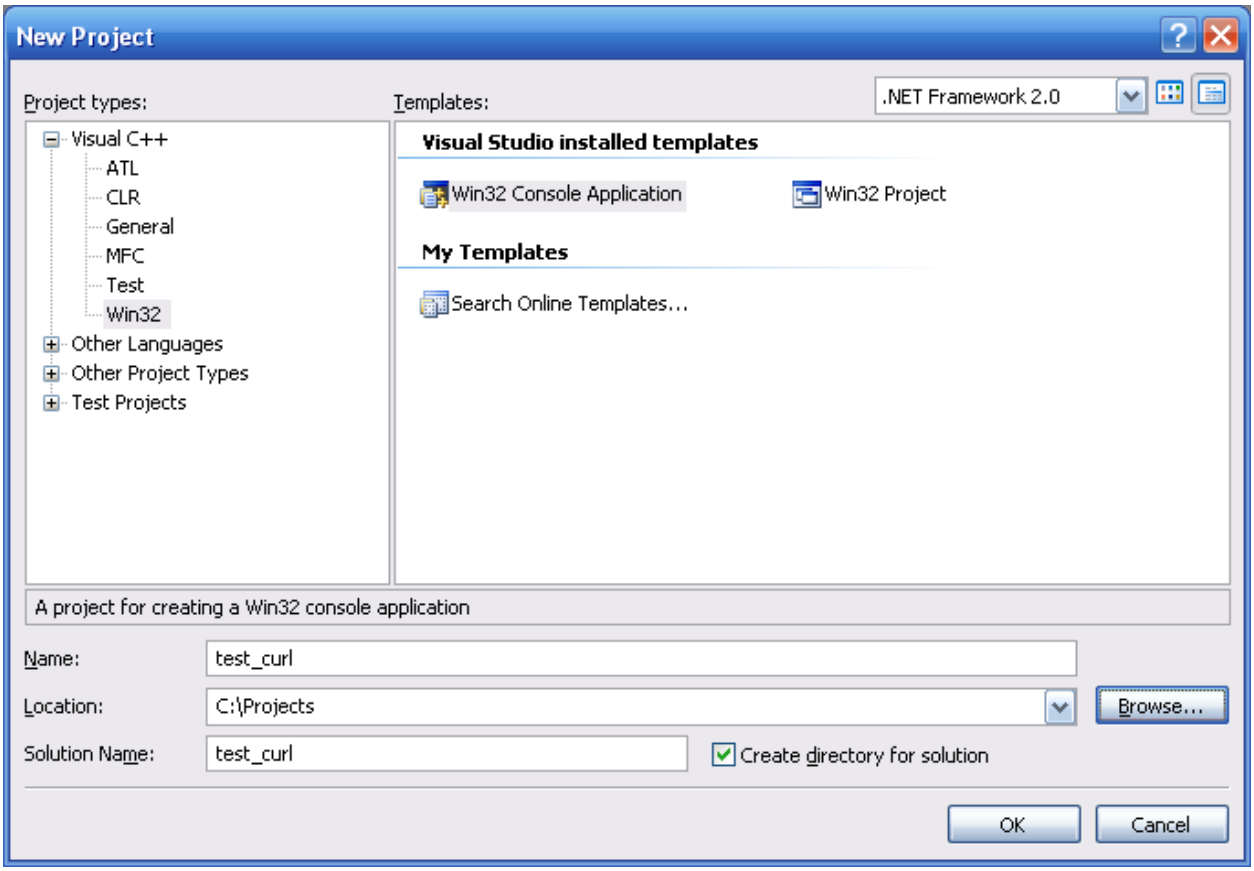
Notice that there are also 2 make files in this directory. We won't need them so they can be safely deleted, if you wish.

Finally, you need to copy *curllib.lib* from *C:\curl-7.19.2\lib\Debug* to *C:\Projects\custom libraries\lib_dbg*.

4.2 Creating the test project

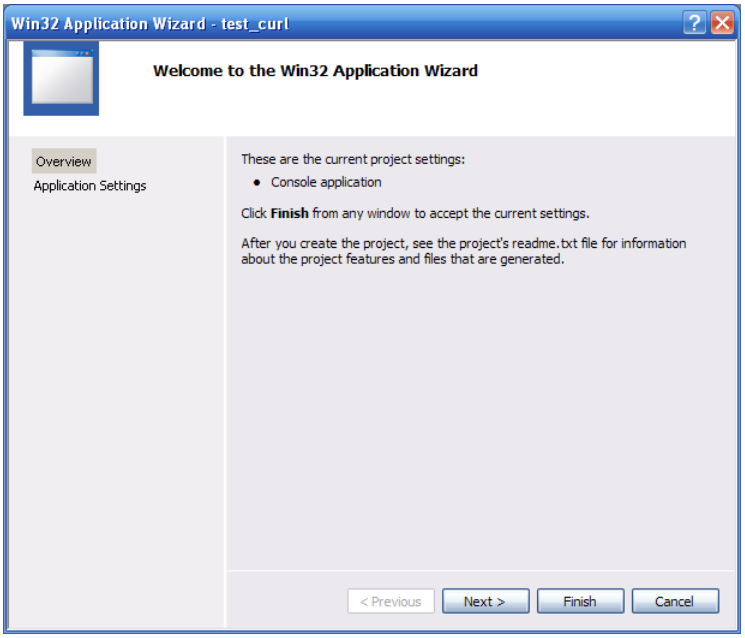
The libcurl library can be used in any type of application. In order to keep things simple, we will create a simple Win32 console application.

1. Fire up VS2008 and go to: **File** -> **New** -> **Project…**
2. First expand the **Visual C++** node, then select the **Win32** node and click on the **Win32 Console Application** item in the **Templates** panel:



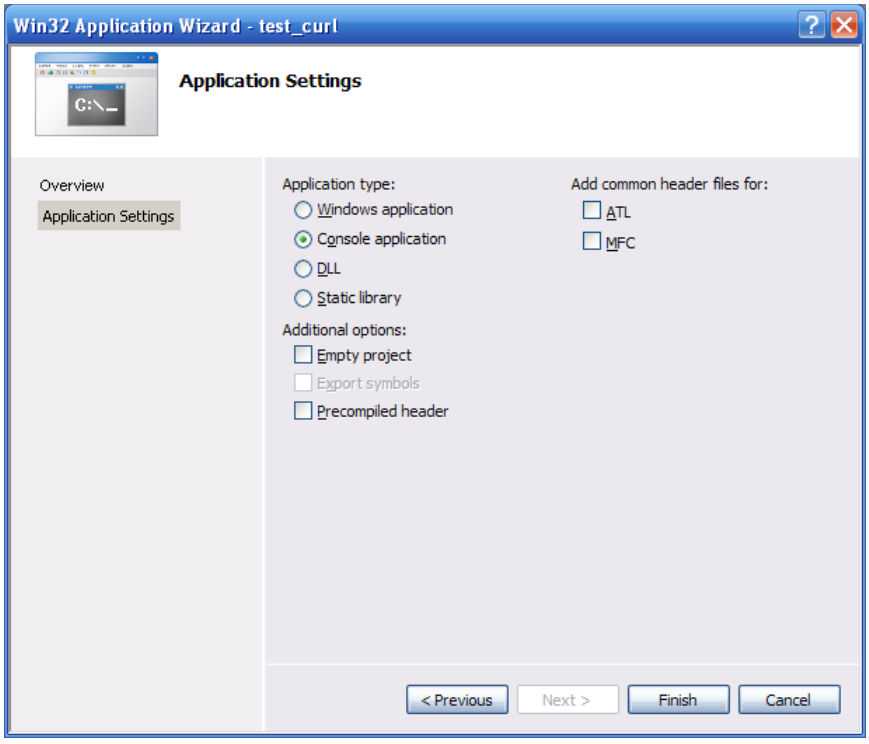
3. Type in “test_curl” for the project name. The project location should be:
   C:\Projects
   Make sure that the checkbox **Create directory for solution** is checked and click on **OK**.

4. The Win32 Application Wizard will appear. In the first window click on **Next**:

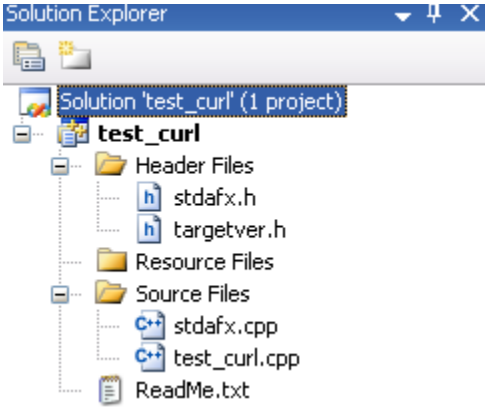


5. In the following window:
    a. Make sure that **Console application** is selected from the **Application type**: list
    b. In **Additional options**, uncheck the **Precompiled header** option.
    c. Ensure that both **ATL** and **MFC** are unchecked in the **Add common header files for:** list.

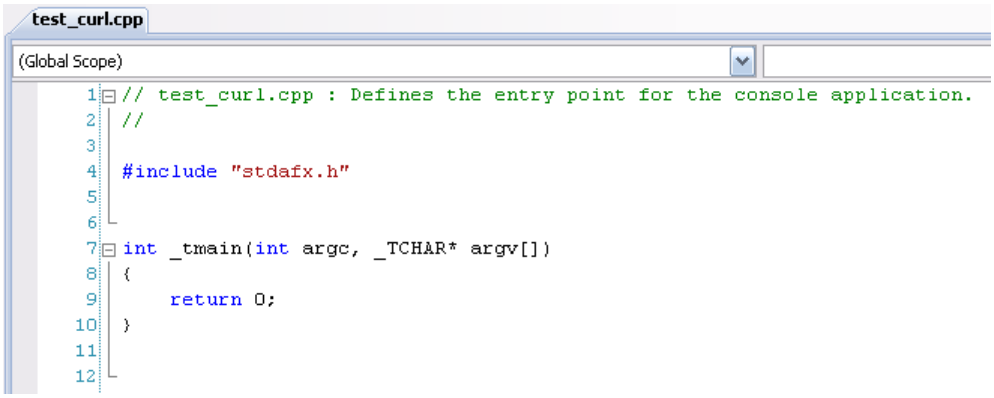
    Now you can press on the **Finish** button and wait for VS2008 to set up your project.



6. In the **Solution Explorer** window you should see the following project structure:



Double-click on test_curl.cpp to open it (if VS2008 didn't already do so for you). The source code should look like this:



4.3 How to use the libcurl library

4.3.1 Sample source code

Replace all the code in *test_curl.cpp* with this code:

```
// headers
#include "stdafx.h"
#include <conio.h>
#include <curl/curl.h>
#include <windows.h>

// prototypes
int         libcurl_progress_callback (void * clientp, double dltotal,
                                       double dlnow, double ultotal,
                                       double ulnow);
size_t      libcurl_read_callback(void * pBuffer, size_t size, size_t nmemb,
                                  void * hFile);
void  SSHUpload(char * strFileName, char * strFilePath);

int _tmain(int argc, _TCHAR* argv[])
{
      SSHUpload("test.txt", "C:\\");
      printf("Press any key to continue...");
      _getch();

      return 0;
}

void SSHUpload(char * strFileName, char * strFilePath)
{
      char strBuffer[1024];
      CURL * hCurl;
      CURLcode ccCurlResult = CURL_LAST;
      curl_off_t cotFileSize;
      HANDLE hFile;
      LARGE_INTEGER liFileSize;

      // check parameters
      if((strFileName == NULL || strlen(strFileName) == 0) ||
         (strFilePath == NULL || strlen(strFilePath) == 0))
            return;

      // parse file path
      if(strFilePath[strlen(strFilePath) - 1] == '\\')
            sprintf_s(strBuffer, 1024, "%s%s", strFilePath, strFileName);
      else
            sprintf_s(strBuffer, 1024, "%s\\%s", strFilePath, strFileName);

      // create a handle to the file
      hFile = CreateFileA(strBuffer,                  // file to open
                          GENERIC_READ,               // open for reading
                          FILE_SHARE_READ,            // share for reading
                          NULL,                       // default security
                          OPEN_EXISTING,              // existing file only
                          FILE_ATTRIBUTE_NORMAL,      // normal file
                          NULL);                      // no attr. template
```

```
	if(hFile != INVALID_HANDLE_VALUE)
	{
		// global libcurl initialisation
		ccCurlResult = curl_global_init(CURL_GLOBAL_WIN32);
		if(ccCurlResult == 0)
		{
			// start libcurl easy session
			hCurl = curl_easy_init();
			if(hCurl)
			{
				// enable verbose operation
				curl_easy_setopt(hCurl, CURLOPT_VERBOSE, TRUE);

				// enable uploading
				curl_easy_setopt(hCurl, CURLOPT_UPLOAD, TRUE);

				// inform libcurl of the file's size
				GetFileSizeEx(hFile, &liFileSize);
				cotFileSize = liFileSize.QuadPart;
				curl_easy_setopt(hCurl,
						CURLOPT_INFILESIZE_LARGE,
						cotFileSize);

				// enable progress report function
				curl_easy_setopt(hCurl, CURLOPT_NOPROGRESS, FALSE);
				curl_easy_setopt(hCurl,
						CURLOPT_PROGRESSFUNCTION,
						libcurl_progress_callback);

				// use custom read function
				curl_easy_setopt(hCurl,
						CURLOPT_READFUNCTION,
						libcurl_read_callback);

				// specify which file to upload
				curl_easy_setopt(hCurl, CURLOPT_READDATA, hFile);

				// specify full path of uploaded file (i.e. server
				// address plus remote path)
				sprintf_s(strBuffer,
					  1024,
					  "sftp://123.123.123.123/home/user/%s",
					  strFileName);
				curl_easy_setopt(hCurl, CURLOPT_URL, strBuffer);

				// set SSH server port
				curl_easy_setopt(hCurl, CURLOPT_PORT, 22);

				// set SSH user name and password in libcurl in this
				// format "user:password"
				curl_easy_setopt(hCurl,
						CURLOPT_USERPWD,
						"user:password");



				// set SSH authentication to user name and password
```

```
                        curl_easy_setopt(hCurl,
                                         CURLOPT_SSH_AUTH_TYPES,
                                         CURLSSH_AUTH_PASSWORD);

                        // execute command
                        ccCurlResult = curl_easy_perform(hCurl);

                        // end libcurl easy session
                        curl_easy_cleanup(hCurl);
                  }
            }

            // release file handle
            CloseHandle(hFile);

            // global libcurl cleanup
            curl_global_cleanup();

            if (ccCurlResult == CURLE_OK)
                  printf("File uploaded successfully.\n");
            else
                  printf("File upload failed. Curl error: %d\n",
                         ccCurlResult);
      }
      else
            printf("File upload failed! Could not open local file");
}

size_t libcurl_read_callback(void * pBuffer, size_t size,
                             size_t nmemb, void * hFile)
{
      DWORD dwNumberOfBytesRead = 0;

      BOOL bResult = ReadFile((HANDLE) hFile, pBuffer, size * nmemb,
                              &dwNumberOfBytesRead, NULL);

      return dwNumberOfBytesRead;
}

int libcurl_progress_callback (void * clientp, double dltotal, double dlnow,
                               double ultotal, double ulnow)
{
      printf("Uploaded: %d / %d\n", (int) ulnow, (int) ultotal);

      return 0;
}
```

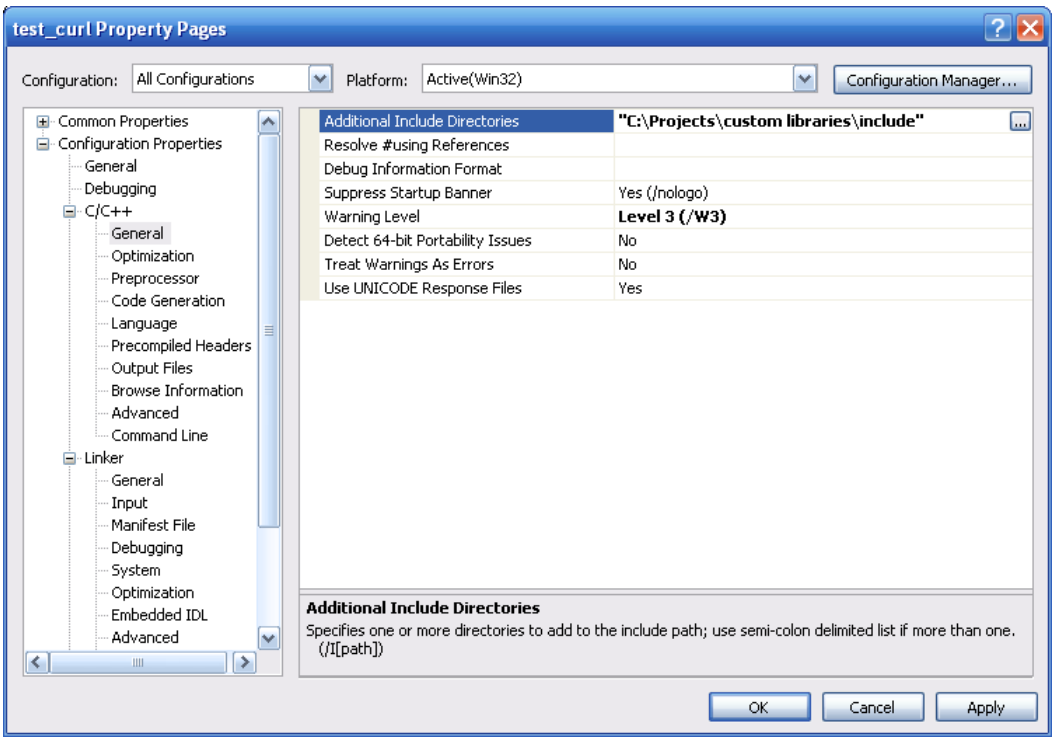
NOTE: You must replace some of the information in the sample code:

- line 17: enter a file name to be uploaded and its location on your computer
- line 93: type in your server's IP address and the remote path where the file should be stored (here I assume that you have access to a computer that is running a SSH server)
- line 104: the login credentials for the SSH server

4.3.2 Adding libcurl to the list of libraries

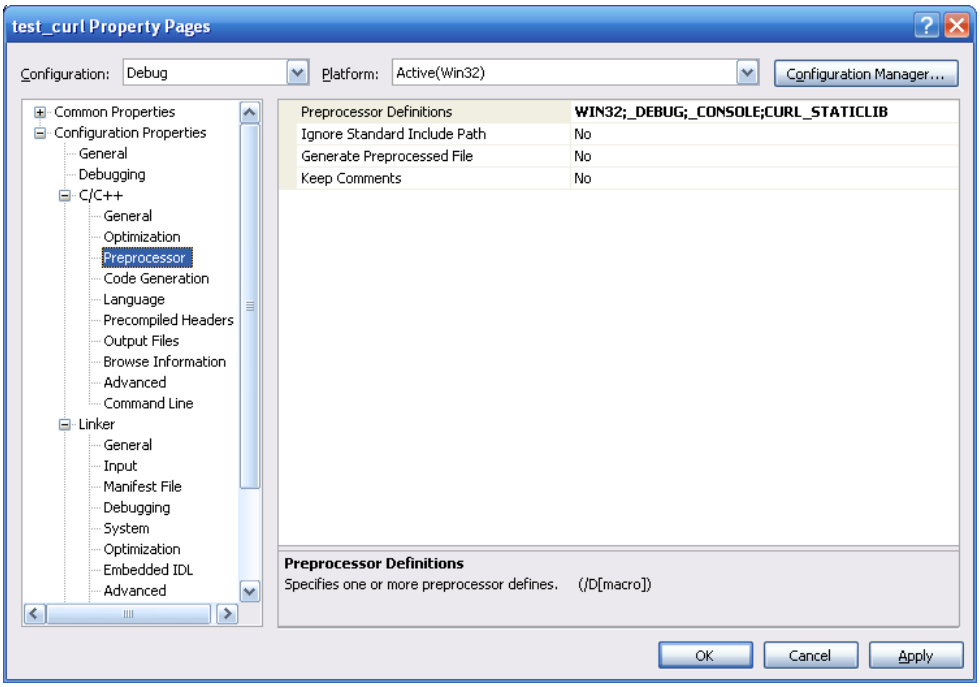
Now let's tell the compiler where to find the libcurl header files and the library itself:

1. In the **Solution Explorer** window, right-click on the test_curl project window and select **Properties** from the pop-up menu.
2. Expand the **Configuration Properties** node.
3. Expand the **C/C++** node
    a. Select the **General** node, choose the **Additional Include Directories** property and add the path:
    "C:\Projects\custom libraries\include"

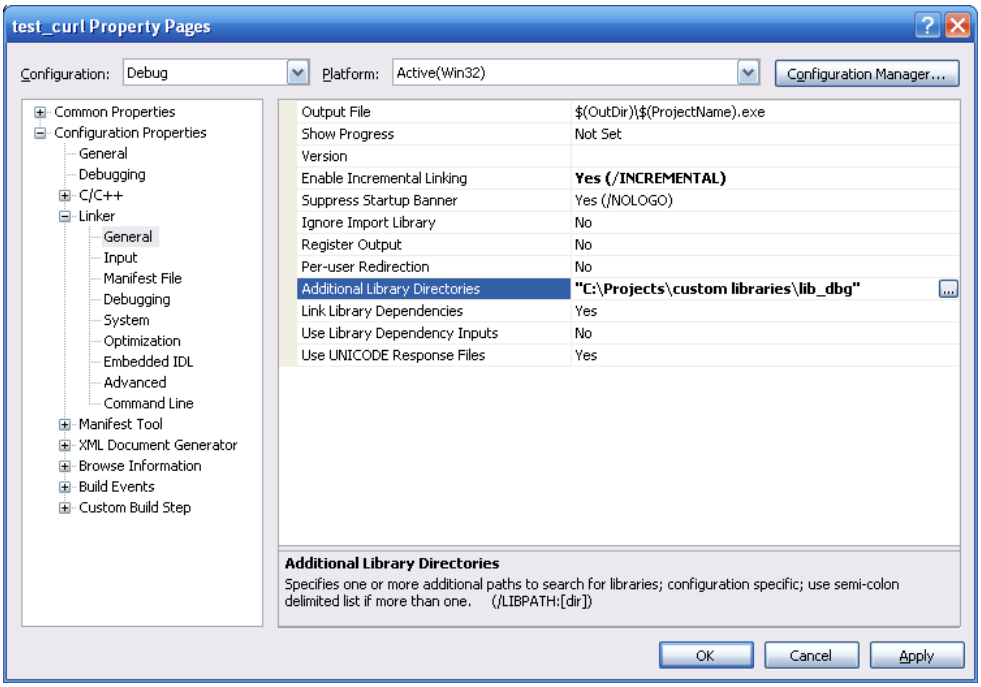


NOTE: While you might be tempted to include the "*C:\Projects\custom libraries\include\curl*" directory instead, we must add the parent directory since some libcurl files use #include statements like this one: "#include <curl/curlbuild.h>".

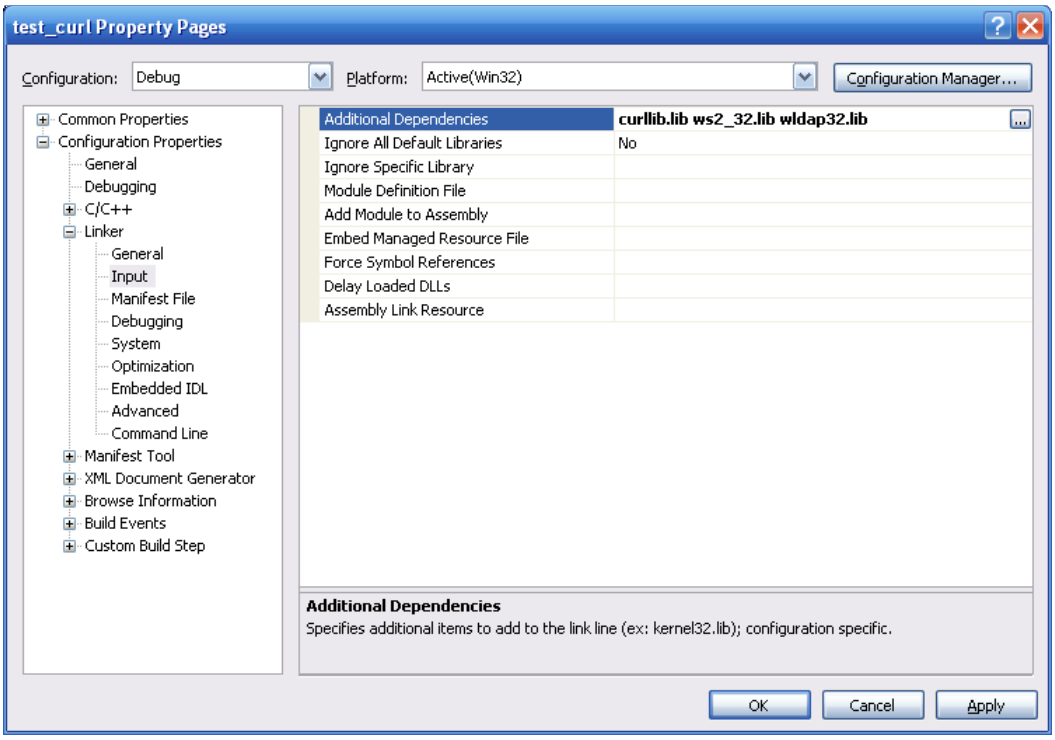
b. Select the **Preprocessor node** and click on the **Preprocessor Definitions** property. Add in the following to the existing definitions:
;CURL_STATICLIB



4. Expand the **Linker** node.
   a. Select the **General** node, chose the **Additional Library Directories** property and add the path:
   "C:\Projects\custom libraries\lib_dbg"



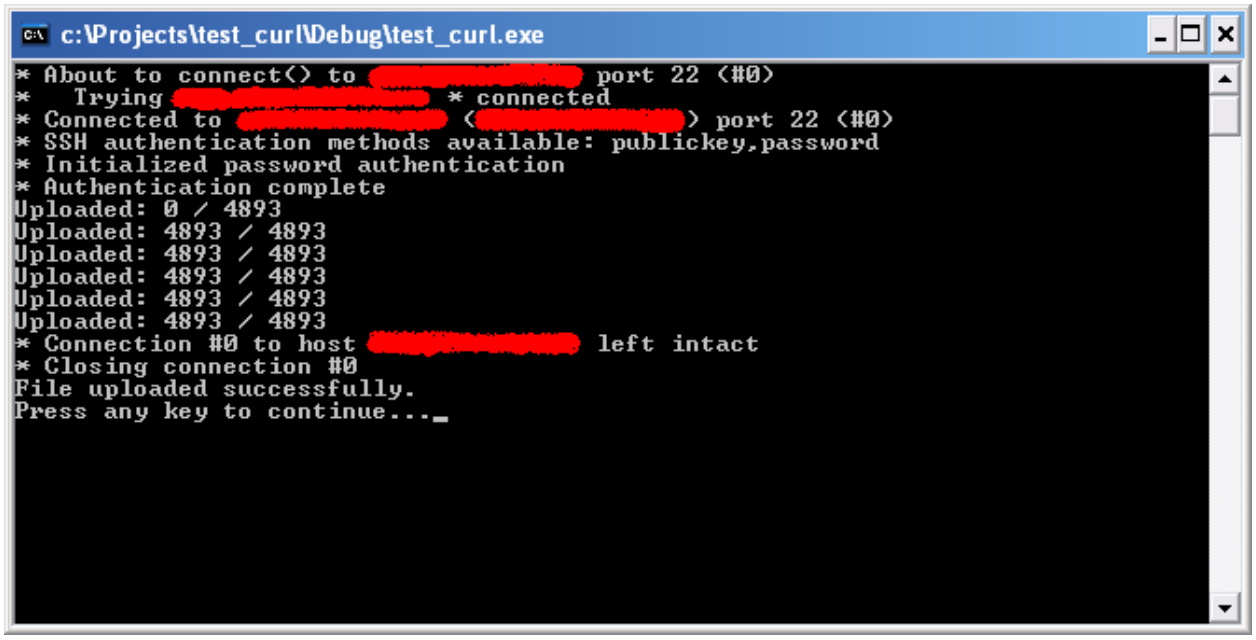
b. Select the **Input** node, click on the **Additional Dependencies** property and set it to:
curllib.lib ws2_32.lib



5. Click on the **OK** button.

4.3.3 The test-drive

Now we are finally ready to reap the fruits of our labor: compile and run the program. Depending on the size of the file you chose to upload, the output should look similar to this:



5. Final Notes

Congratulations! You are now ready to use libcurl in your own projects. If you run into trouble down the road, don't hesitate to post a message on libcurl's very active mailing list: http://cool.haxx.se/mailman/listinfo/curl-library. However, please take a moment and read the mailing list etiquette (http://curl.haxx.se/mail/etiquette.html) before posting.