Using libcurl with SSH support in Visual Studio 2010

Version 1.3

© Andrei Jakab (andrei.jakab@tut.fi)

Revision History

Revision	Date	By	Comment
1.0	11 Jul 2011	Andrei Jakab	Initial version
1.1	17 Jul 2012	Andrei Jakab	Updated the guide to reflect the latest versions of: o ActivePerl (5.14.2.1402) o libcurl (7.26.0) o libSSH2 (1.4.2) o NASM (2.10.01) o OpenSSL (1.0.1c)
1.2	22 Sep 2012	Andrei Jakab	Appendix A: corrected preprocessor definitions
1.3	05 Feb 2013	Andrei Jakab	Updated the guide to reflect the latest versions of: o ActivePerl (5.16.2.1602) o libcurl (7.28.1) o libSSH2 (1.4.3) o NASM (2.10.07)

Acknowledgements

I would like to thank the following people for helping me make this guide what it is today by letting me know about typos and by suggesting improvements:

- Chengwei Lin
- Jon Woellhaf
- Philipp Leusmann
- Reinhard Gentz
- Jack Schmidt
- Huu Minh Nguyen

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
- **<u>red, bold and underlined text</u>** 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 2010.

This document is based on my previous guide “Using libcurl with SSH support in Visual Studio 2008”, which was 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. OpenSSL also requires the Netwide Assembler to be present on your computer.

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.

In order to obtain the latest version of the Netwide Assembler, click on the link in the table below and, on the webpage that appears, follow the link that takes you to the latest stable version of NASM. You have the choice between downloading the source code or pre-compiled binaries. I strongly encourage you to download the binary files since I will not cover the compilation of NASM in this guide. The archives that contain the binary distributions are located in folders named after their intended architecture e.g. win32, dos, os2 etc.

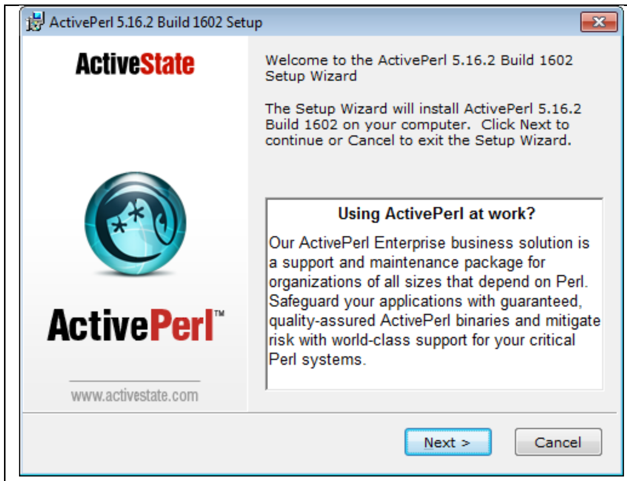
Software	URL	Current Version
ActivePerl	http://www.activestate.com/activeperl/downloads	5.16.2.1602
OpenSSL	http://www.openssl.org/source	1.0.1c
libSSH2	http://www.libssh2.org	1.4.3
libcurl	http://curl.haxx.se/download.html	7.28.1
Netwide Assembler	http://www.nasm.us	2.10.07
7-Zip	http://www.7-zip.org	9.20

NOTE: This guide assumes that you already have successfully installed Microsoft Visual Studio 2010 (VS2010). The examples in this document were built using version 10.0.40219.1 SP1 of VS2010.

3. Installation / Compilation

3.1 ActivePerl

Installing ActivePerl is fairly straightforward as long as you are logged in with an Administrator account. 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.



ActivePerl 5.16.2 Build 1602 Setup

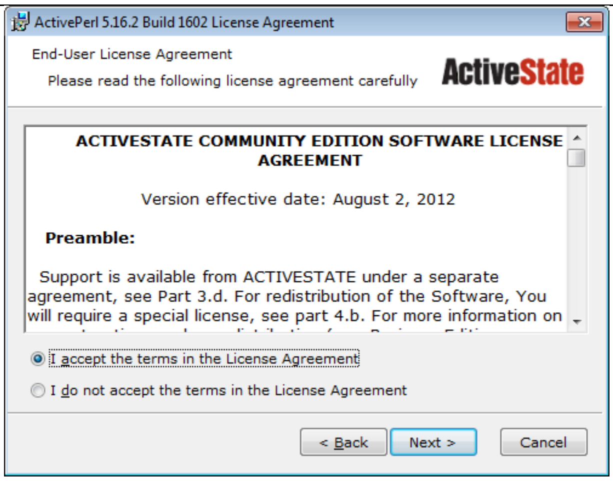
Welcome to the ActivePerl 5.16.2 Build 1602 Setup Wizard

The Setup Wizard will install ActivePerl 5.16.2 Build 1602 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Using ActivePerl at work?

Our ActivePerl Enterprise business solution is a support and maintenance package for organizations of all sizes that depend on Perl. Safeguard your applications with guaranteed, quality-assured ActivePerl binaries and mitigate risk with world-class support for your critical Perl systems.

Step 1 - Introduction



ActivePerl 5.16.2 Build 1602 License Agreement

End-User License Agreement

Please read the following license agreement carefully

ACTIVESTATE COMMUNITY EDITION SOFTWARE LICENSE AGREEMENT

Version effective date: August 2, 2012

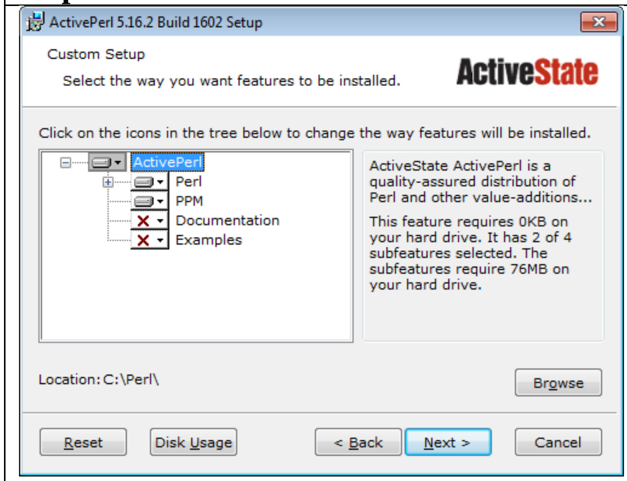
Preamble:

Support is available from ACTIVESTATE under a separate agreement, see Part 3.d. For redistribution of the Software, You will require a special license, see part 4.b. For more information on

● I accept the terms in the License Agreement

○ I do not accept the terms in the License Agreement

Step 2 - EULA



ActivePerl 5.16.2 Build 1602 Setup

Custom Setup

Select the way you want features to be installed.

Click on the icons in the tree below to change the way features will be installed.

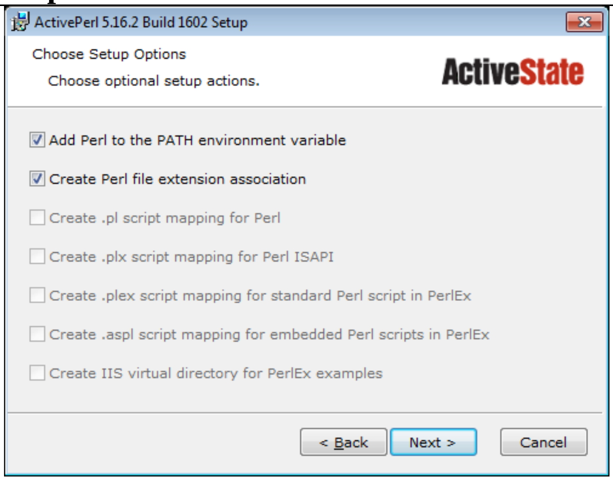
- ActivePerl
  - Perl
  - PPM
  - ✗ Documentation
  - ✗ Examples

ActiveState ActivePerl is a quality-assured distribution of Perl and other value-additions...

This feature requires 0KB on your hard drive. It has 2 of 4 subfeatures selected. The subfeatures require 76MB on your hard drive.

Location: C:\Perl\

Step 3 - Customization 1



ActivePerl 5.16.2 Build 1602 Setup

Choose Setup Options

Choose optional setup actions.

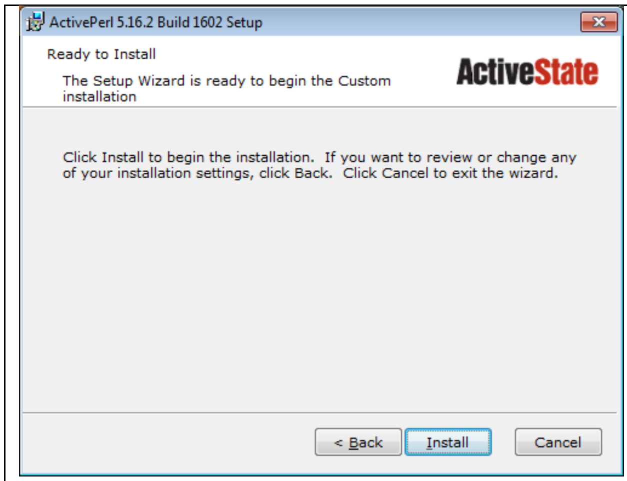
- ☑ Add Perl to the PATH environment variable
- ☑ Create Perl file extension association
- ☐ Create .pl script mapping for Perl
- ☐ Create .plx script mapping for Perl ISAPI
- ☐ Create .plex script mapping for standard Perl script in PerlEx
- ☐ Create .aspl script mapping for embedded Perl scripts in PerlEx
- ☐ Create IIS virtual directory for PerlEx examples

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 folder. Make sure to change this if you like to keep your C: folder tidy.

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

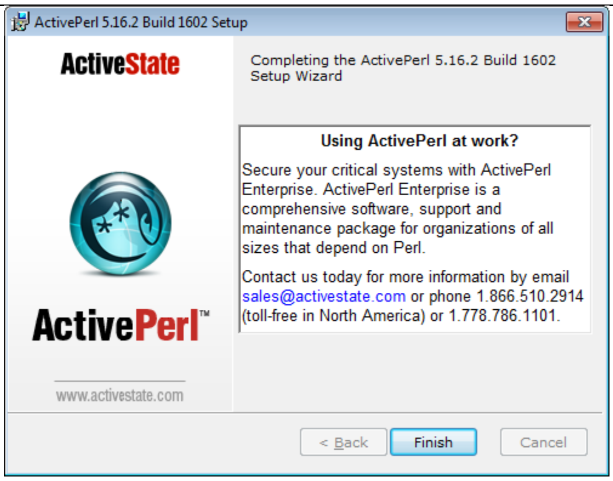

ActivePerl 5.16.2 Build 1602 Setup

Ready to Install

The Setup Wizard is ready to begin the Custom installation

Click Install to begin the installation. If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Step 5 - Installation Confirmation


ActivePerl 5.16.2 Build 1602 Setup

Completing the ActivePerl 5.16.2 Build 1602 Setup Wizard

Using ActivePerl at work?

Secure your critical systems with ActivePerl Enterprise. ActivePerl Enterprise is a comprehensive software, support and maintenance package for organizations of all sizes that depend on Perl.

Contact us today for more information by email sales@activestate.com or phone 1.866.510.2914 (toll-free in North America) or 1.778.786.1101.

Step 6 - Installation Complete

3.2 Netwide Assembler

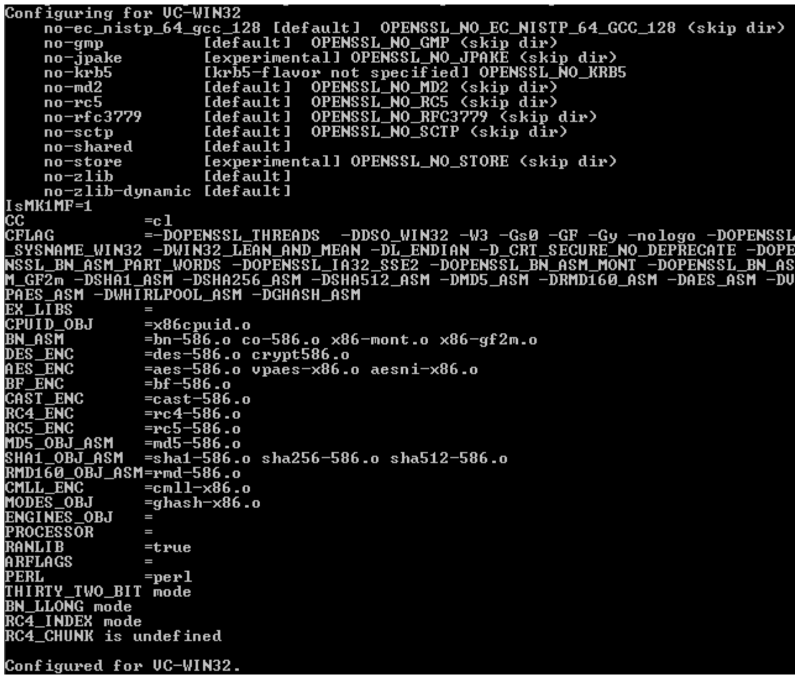
1. Extract the *nasm-2.10.07* folder from the zip file and place it in the C: root directory

3.3 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-1.0.1c.tar* file from the gzip file to a temporary directory
2. Extract the *openssl-1.0.1c* folder from the tar file and place it in the C: root directory
3. Close the VS2010 IDE and open a Visual Studio 2010 command prompt. If you've installed VS 2010 with the default settings, the command prompt shortcut should be located in **Start** -> **All Programs** -> **Microsoft Visual Studio 2010** -> **Visual Studio Tools** -> **Visual Studio Command Prompt (2010)**
4. Add the NASM executable to the PATH environment variable: `path = %PATH%;C:\nasm-2.10.07`
5. Create the directory where the output of the compilation process will be stored: `mkdir c:\openssl_lib`
6. Change the working directory to the OpenSSL directory: `cd /D c:\openssl-1.0.1c`
7. 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:



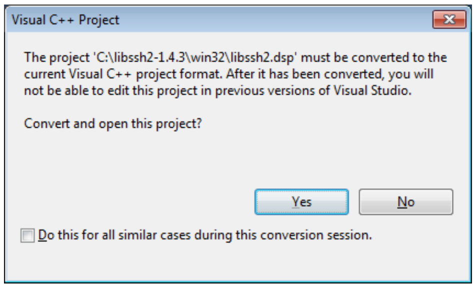
```
Configuring for VC-WIN32
    no-ec_nistp_64_gcc_128 [default]  OPENSSL_NO_EC_NISTP_64_GCC_128 (skip dir)
    no-gmp          [default]  OPENSSL_NO_GMP (skip dir)
    no-jpake        [experimental] OPENSSL_NO_JPAKE (skip dir)
    no-krb5         [krb5-flavor not specified] OPENSSL_NO_KRB5
    no-md2          [default]  OPENSSL_NO_MD2 (skip dir)
    no-rc5          [default]  OPENSSL_NO_RC5 (skip dir)
    no-rfc3779      [default]  OPENSSL_NO_RFC3779 (skip dir)
    no-sctp         [default]  OPENSSL_NO_SCTP (skip dir)
    no-shared       [default]
    no-store        [experimental] OPENSSL_NO_STORE (skip dir)
    no-zlib         [default]
    no-zlib-dynamic [default]
IsMK1MF=1
CC            =cl
CFLAG         =-DOPENSSL_THREADS  -DDSO_WIN32 -W3 -Gs0 -GF -Gy -nologo -DOPENSSL
_SYSNAME_WIN32 -DWIN32_LEAN_AND_MEAN -DL_ENDIAN -D_CRT_SECURE_NO_DEPRECATE -DOPE
NSSL_BN_ASM_PART_WORDS -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_AS
M_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM -DV
PAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM
EX_LIBS       =
CPUID_OBJ     =x86cpuid.o
BN_ASM        =bn-586.o co-586.o x86-mont.o x86-gf2m.o
DES_ENC       =des-586.o crypt586.o
AES_ENC       =aes-586.o vpaes-x86.o aesni-x86.o
BF_ENC        =bf-586.o
CAST_ENC      =cast-586.o
RC4_ENC       =rc4-586.o
RC5_ENC       =rc5-586.o
MD5_OBJ_ASM   =md5-586.o
SHA1_OBJ_ASM  =sha1-586.o sha256-586.o sha512-586.o
RMD160_OBJ_ASM=rmd-586.o
CMLL_ENC      =cmll-x86.o
MODES_OBJ     =ghash-x86.o
ENGINES_OBJ   =
PROCESSOR     =
RANLIB        =true
ARFLAGS       =
PERL          =perl
THIRTY_TWO_BIT mode
BN_LLONG mode
RC4_INDEX mode
RC4_CHUNK is undefined

Configured for VC-WIN32.
```

8. Create the required assembly files:
   `ms\do_nasm`
   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`
9. Compile the static library:
   `nmake -f ms\nt.mak`
10. The compilation process takes a while so you can go grab some coffee/tea at this point.
11. If all is well, at the end of the compilation you will have some libraries and a number of executables in *C:\openssl-1.0.1c\out32*
12. 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 this command finishes executing.
13. To install OpenSSL to the location you specified in step 6, run:
    `nmake -f ms\nt.mak install`

Note: You can find additional compiling instructions in *C:\openssl-1.0.1c\INSTALL.W32*; this file also contains a troubleshooting section that could help you out if something goes wrong during the compilation process.

3.4 libSSH2

1. Extract the *libssh2-1.4.3.tar* file from the gzip file to a temporary directory.
2. Extract the *libssh2-1.4.3* directory from the tar file and place it in the C: root directory.
3. Open *C:\libssh2-1.4.3\win32\libssh2.dsp* in the Visual Studio 2010 IDE. Since this project file was created using an older version of Visual Studio, the following message box will appear:

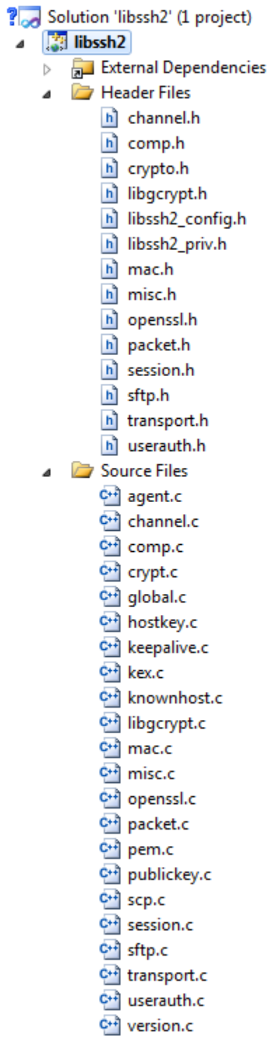

Visual C++ Project

The project 'C:\libssh2-1.4.3\win32\libssh2.dsp' must be converted to the current Visual C++ project format. After it has been converted, you will not be able to edit this project in previous versions of Visual Studio.

Convert and open this project?

Yes
No
Do this for all similar cases during this conversion session.

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


Solution 'libssh2' (1 project)
libssh2
External Dependencies
Header Files
channel.h
comp.h
crypto.h
libgcrypt.h
libssh2_config.h
libssh2_priv.h
mac.h
misc.h
openssl.h
packet.h
session.h
sftp.h
transport.h
userauth.h
Source Files
agent.c
channel.c
comp.c
crypt.c
global.c
hostkey.c
keepalive.c
kex.c
knownhost.c
libgcrypt.c
mac.c
misc.c
openssl.c
packet.c
pem.c
publickey.c
scp.c
session.c
sftp.c
transport.c
userauth.c
version.c

4. Set the LIB Debug as the active solution configuration. On the **Build** menu in Visual Studio, click **Configuration Manager…** and in the window that appears select **LIB Debug** from the **Active solution configuration** drop-down list. Close the Configuration Manager.

5. Next, you must tell the compiler and the librarian where the OpenSSL library is located and how to compile the library:
   a. Right-click on the libssh2 project in the **Solution Explorer** window and select **Properties** from the pop-up menu.
   b. Expand the **Configuration Properties** node, click on the **C/C++** node
      i. Select **General**, choose the **Additional Include Directories** property and add the following:
         ;C:\openssl_lib\include
         (the semicolon needed since other paths are already present)
      ii. Next, click on **Code Generation** and in the **Runtime Library** field, select the **Multi-threaded Debug DLL (/MDd)** option.
   c. Also 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.

6. Now we are ready to compile. Right-click on the libssh2 project and select **Build** from the pop-up menu. VS2010 will prompt you to save the solution file that was created for this project. Once you save it, compilation will begin.

7. The compiler might display a couple of security warnings and/or "possible loss of data" warnings. Also the linker might display a number of LNK4221 warnings. For our purposes, both of these types of warnings can be safely ignored.

3.5 libcurl

1. Extract the *curl-7.28.1* 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-7.28.1\lib\libcurl.vcproj* in the Visual Studio 2010 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 VS2010 format but will generate a couple of warnings, which can be safely ignored.

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 libcurl 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**, add the following:
         ;C:\libssh2-1.4.3\include
         (the semicolon is needed since there are already two additional include directories 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:\libssh2-1.4.3\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. VS2010 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: libcurl, Configuration: Debug Win32 ------
1>  wildcard.c
1>  warnless.c
1>  version.c
1>  url.c
1>  transfer.c
```

```
1>  timeval.c
…
1>  curl_darwinssl.c
1>  curl_addrinfo.c
1>  cookie.c
1>  content_encoding.c
1>  Generating Code...
1>  Compiling...
1>  connect.c
1>  base64.c
1>  axtls.c
1>  asyn-thread.c
1>  asyn-ares.c
1>  amigaos.c
1>  Generating Code...
1>libssh2.lib(ecp_nistputil.obj) : warning LNK4221: This object file does not define any previously undefined public symbols, so it will not be used by any link operation that consumes this library
1>libssh2.lib(ecp_nistp521.obj) : warning LNK4221: This object file does not define any previously undefined public symbols, so it will not be used by any link operation that consumes this library
1>libssh2.lib(ecp_nistp256.obj) : warning LNK4221: This object file does not define any previously undefined public symbols, so it will not be used by any link operation that consumes this library
1>libssh2.lib(ecp_nistp224.obj) : warning LNK4221: This object file does not define any previously undefined public symbols, so it will not be used by any link operation that consumes this library
1>libssh2.lib(fips_ers.obj) : warning LNK4221: This object file does not define any previously undefined public symbols, so it will not be used by any link operation that consumes this library
1>  libcurl.vcxproj -> C:\curl-7.28.1\lib\.\Debug\libcurl.lib
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

The LNK4221 warnings can be safely ignored.

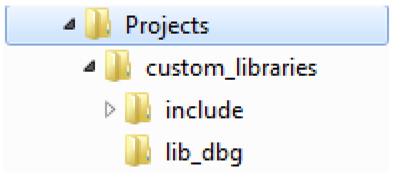
By default, the output directory is *C:\curl-7.28.1\lib\Debug*. In order to create an application that uses the libcurl library, we only need the *libcurl.lib* file from the output directory and the *C:\curl-7.28.1\include\curl* directory, which contains the library's header files.

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.

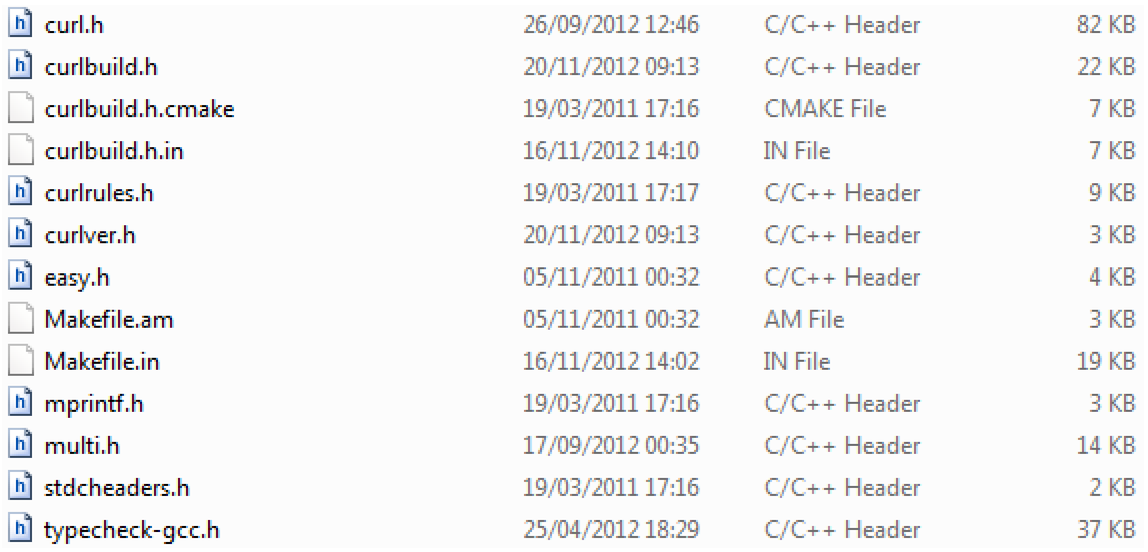
4.1 Preparing the project's file structure

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



```
Projects
  custom_libraries
    include
    lib_dbg
```

Copy the *curl* folder from the *C:\curl-7.28.1\include* directory into the *C:\Projects\custom libraries\include* directory. The *curl* directory should contain the following files:



curl.h	26/09/2012 12:46	C/C++ Header	82 KB
curlbuild.h	20/11/2012 09:13	C/C++ Header	22 KB
curlbuild.h.cmake	19/03/2011 17:16	CMAKE File	7 KB
curlbuild.h.in	16/11/2012 14:10	IN File	7 KB
curlrules.h	19/03/2011 17:17	C/C++ Header	9 KB
curlver.h	20/11/2012 09:13	C/C++ Header	3 KB
easy.h	05/11/2011 00:32	C/C++ Header	4 KB
Makefile.am	05/11/2011 00:32	AM File	3 KB
Makefile.in	16/11/2012 14:02	IN File	19 KB
mprintf.h	19/03/2011 17:16	C/C++ Header	3 KB
multi.h	17/09/2012 00:35	C/C++ Header	14 KB
stdcheaders.h	19/03/2011 17:16	C/C++ Header	2 KB
typecheck-gcc.h	25/04/2012 18:29	C/C++ Header	37 KB

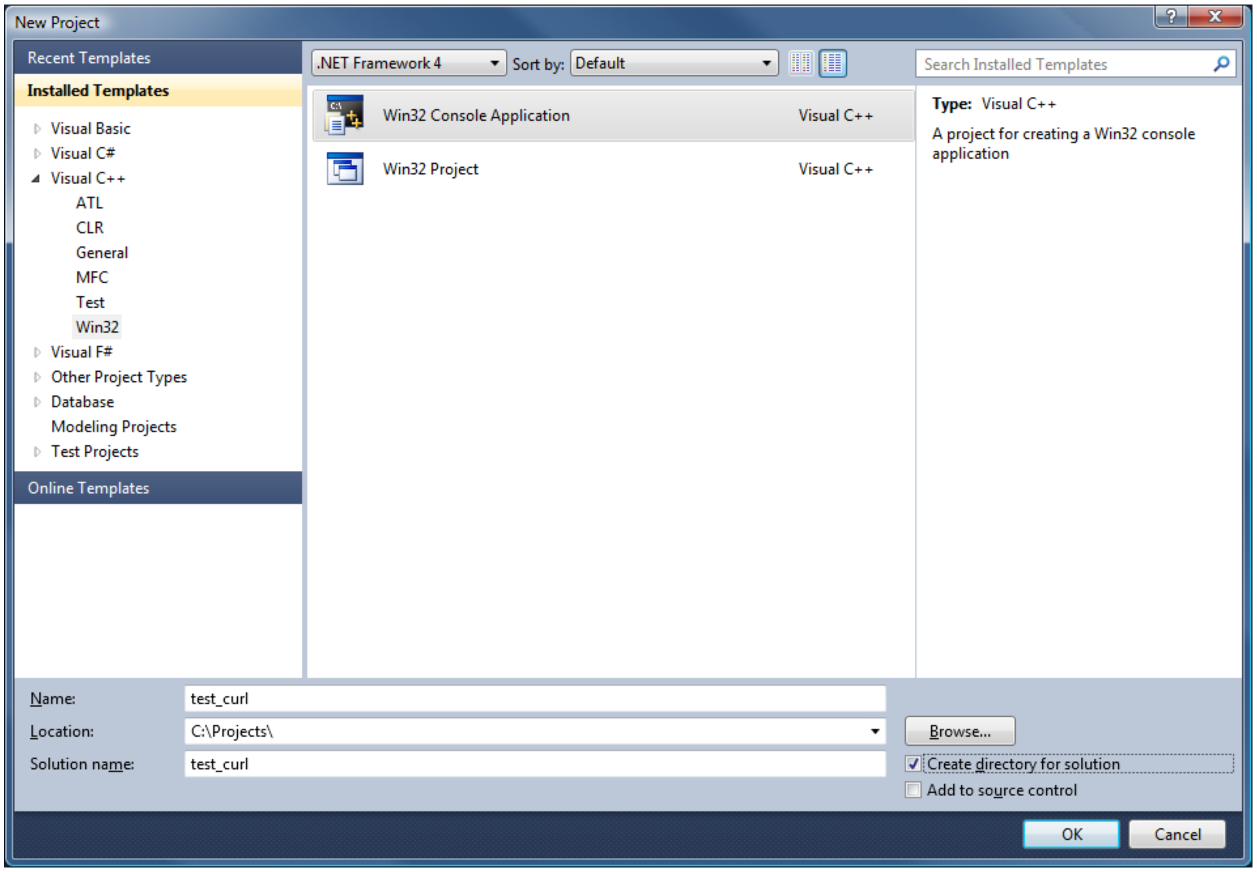
Notice that there are also 4 make files in this directory. We won't need them so they can be safely deleted, if you wish.

Finally, you need to copy *liburl.lib* from *C:\curl-7.28.1\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 VS2010 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 middle panel:



Type: Visual C++

A project for creating a Win32 console application

Name: test_curl

Location: C:\Projects\

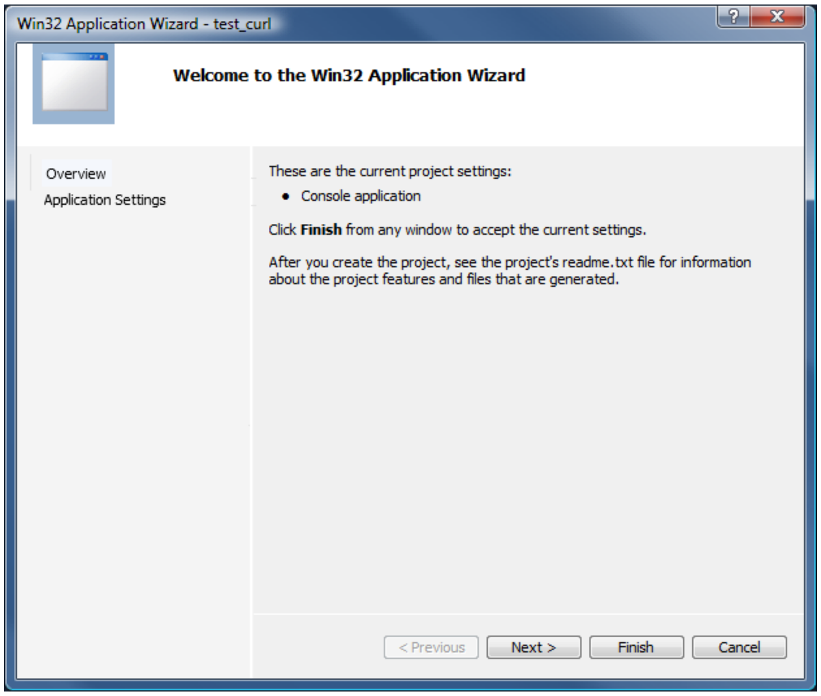
Solution name: test_curl

[✓] Create directory for solution

[ ] Add to source control

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**:



Win32 Application Wizard - test_curl

Welcome to the Win32 Application Wizard

These are the current project settings:

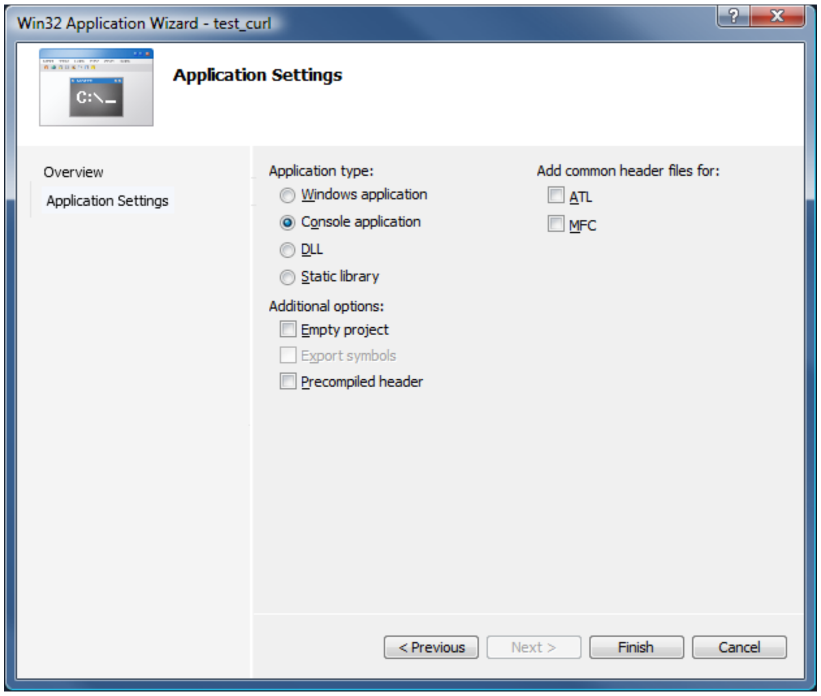
- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

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 VS2010 to set up your project.



Win32 Application Wizard - test_curl

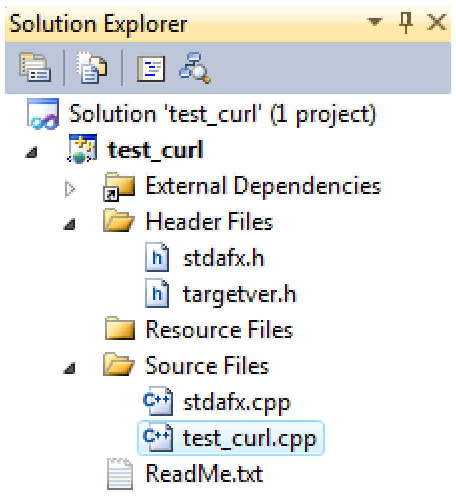
Application Settings

Application type:
( ) Windows application
[✓ Console application]
( ) DLL
( ) Static library

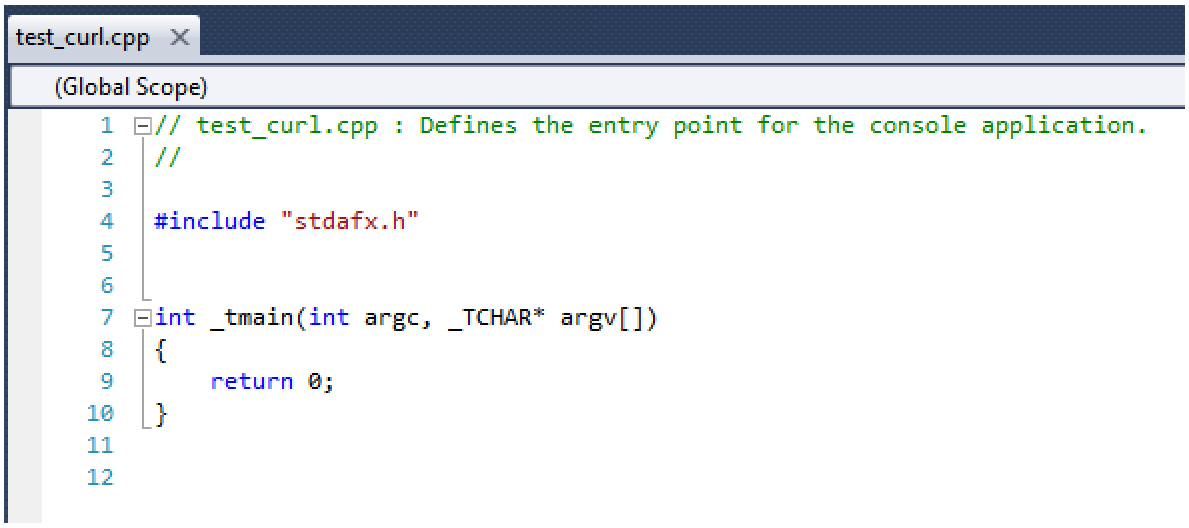
Additional options:
☐ Empty project
☐ Export symbols
☐ Precompiled header

Add common header files for:
☐ ATL
☐ MFC

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



Double-click on test_curl.cpp to open it (if VS2010 didn't already do so for you). The source code should look like this:



```
// test_curl.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}


```

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;
}
```

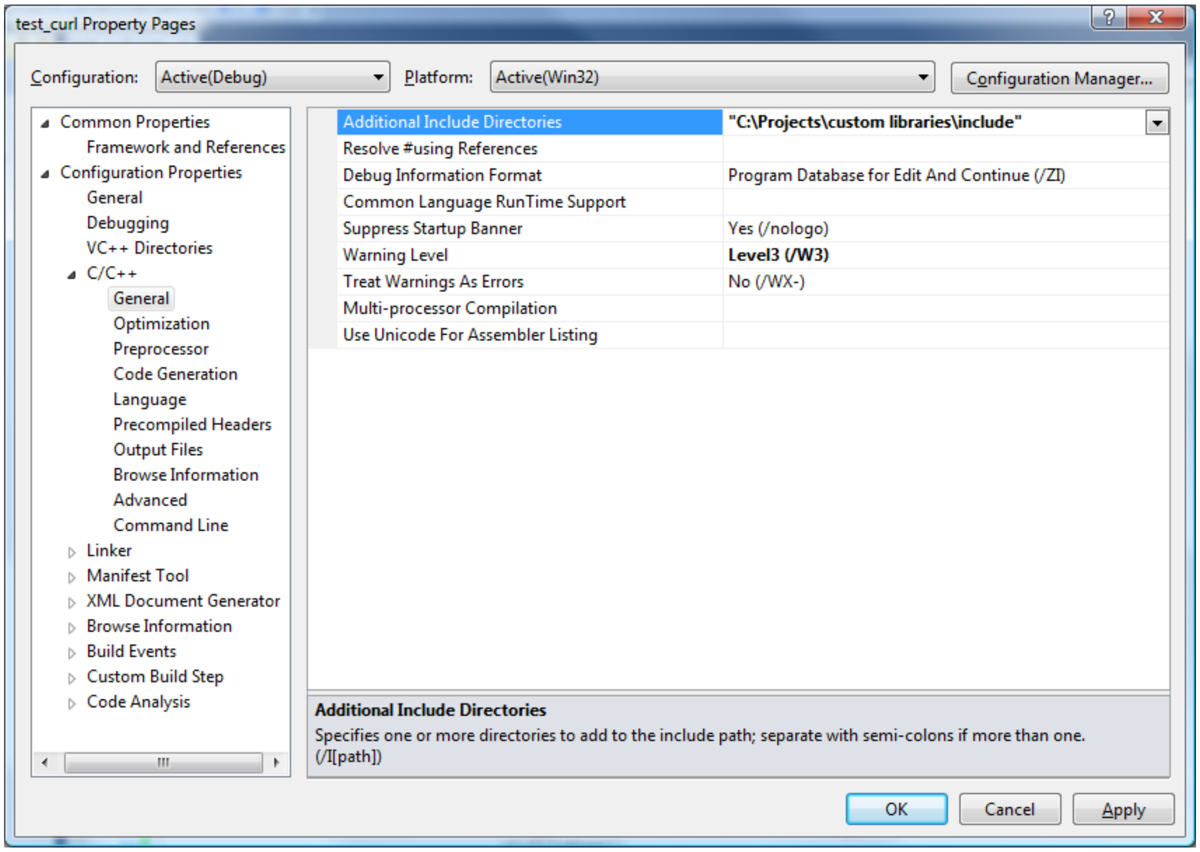
NOTE: You must replace some of the information in the sample code:

- 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

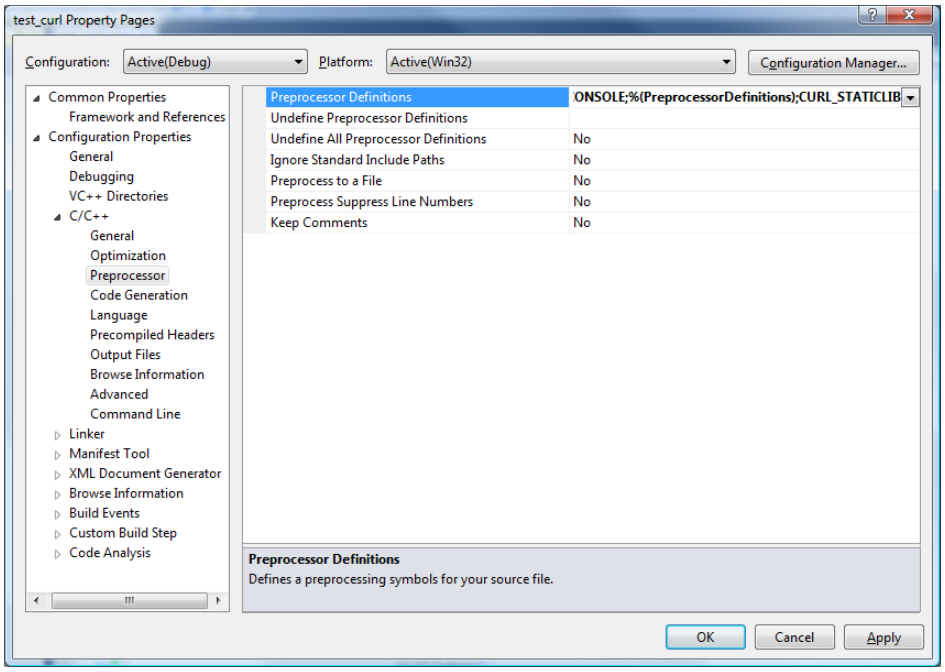


Additional Include Directories	"C:\Projects\custom libraries\include"
Resolve #using References	
Debug Information Format	Program Database for Edit And Continue (/ZI)
Common Language RunTime Support	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level3 (/W3)
Treat Warnings As Errors	No (/WX-)
Multi-processor Compilation	
Use Unicode For Assembler Listing	

Additional Include Directories
Specifies one or more directories to add to the include path; separate with semi-colons if more than one. (/I[path])

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>".

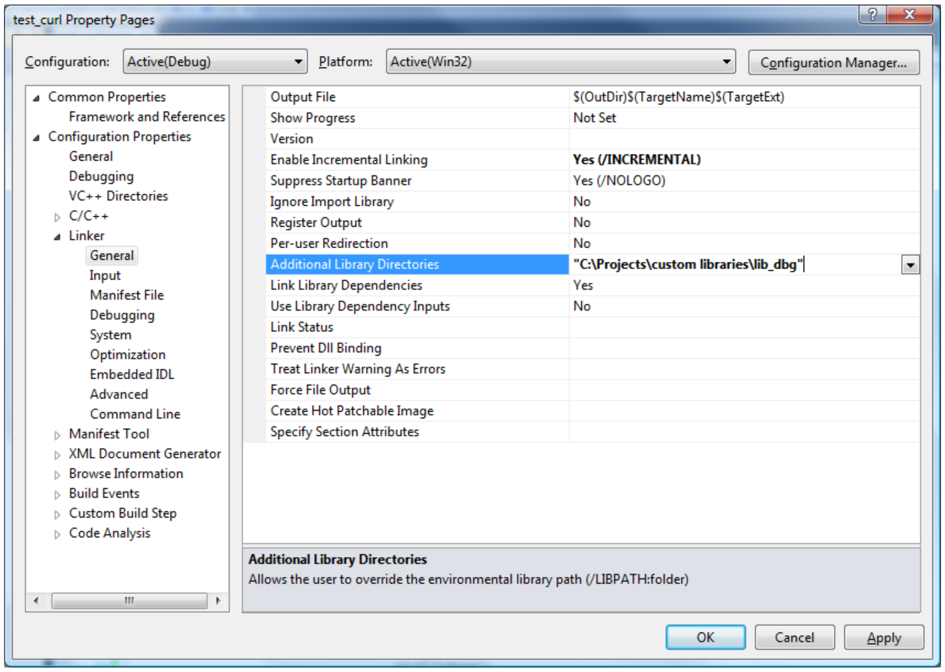
b. Select the **Preprocessor node** and click on the **Preprocessor Definitions** property. Add in the following to the existing definitions:
;CURL_STATICLIB



Preprocessor Definitions	ONSOLE;%(PreprocessorDefinitions);CURL_STATICLIB
Undefine Preprocessor Definitions	
Undefine All Preprocessor Definitions	No
Ignore Standard Include Paths	No
Preprocess to a File	No
Preprocess Suppress Line Numbers	No
Keep Comments	No

Preprocessor Definitions
Defines a preprocessing symbols for your source file.

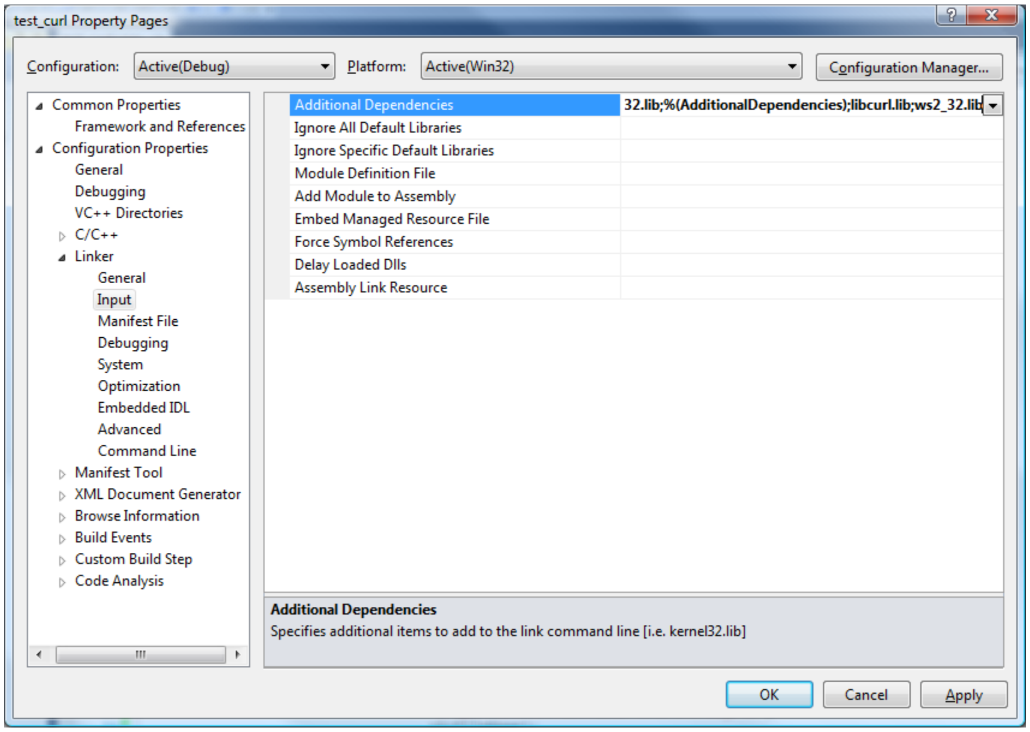
4. Expand the **Linker** node.
   a. Select the **General** node, chose the **Additional Library Directories** property and add the path:
   C:\Projects\custom_libraries\lib_dbg



Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	**Yes (/INCREMENTAL)**
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	"C:\Projects\custom libraries\lib_dbg"
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

Additional Library Directories
Allows the user to override the environmental library path (/LIBPATH:folder)

b. Select the **Input** node, click on the **Additional Dependencies** property and add the following to the existing list of libraries:
;libcurl.lib;ws2_32.lib


test_curl Property Pages

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

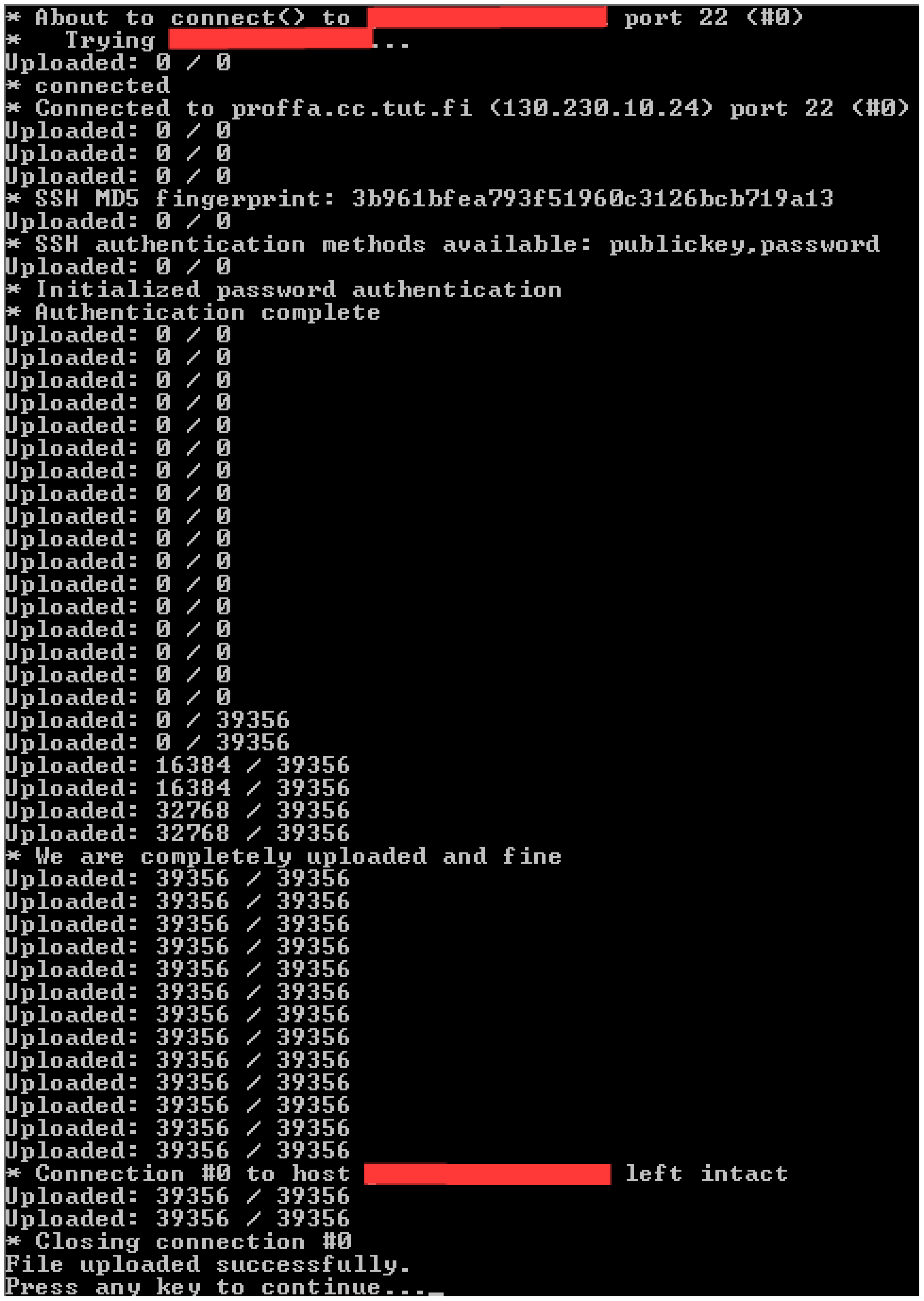
Additional Dependencies	32.lib;%(AdditionalDependencies);libcurl.lib;ws2_32.lib
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

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

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 the figure on the next page.



```
* About to connect() to [redacted] port 22 (#0)
*   Trying [redacted]...
Uploaded: 0 / 0
* connected
* Connected to proffa.cc.tut.fi (130.230.10.24) port 22 (#0)
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
* SSH MD5 fingerprint: 3b961bfea793f51960c3126bcb719a13
Uploaded: 0 / 0
* SSH authentication methods available: publickey,password
Uploaded: 0 / 0
* Initialized password authentication
* Authentication complete
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 0
Uploaded: 0 / 39356
Uploaded: 0 / 39356
Uploaded: 16384 / 39356
Uploaded: 16384 / 39356
Uploaded: 32768 / 39356
Uploaded: 32768 / 39356
* We are completely uploaded and fine
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
* Connection #0 to host [redacted] left intact
Uploaded: 39356 / 39356
Uploaded: 39356 / 39356
* Closing connection #0
File uploaded successfully.
Press any key to continue..._
```

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.

Appendix A – Adding OpenSSL support directly to libcurl

Some of you might find it useful to have SSL/TLS support directly built into libcurl. In this case, compilation procedure that was presented in section 3.5 is identical with the exception of step 3, which must be replaced with the following:

3. Next, you must tell the compiler and the librarian to use the OpenSSL library and where this library is located:
    a. Right-click on the libcurl 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**, add the following:
        ;C:\libssh2-1.4.3\include;c:\openssl_lib\include\;c:\openssl_lib\include\openssl
        (the semicolon is needed since there are already two additional include directories 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;USE_SSLEAY;USE_OPENSSL
    c. In the **Configuration Properties** node, expand the **Librarian** node, and select **General**.
        i. Select **Additional Dependencies** and type in:
        libssh2.lib;libeay32.lib;ssleay32.lib
        ii. Next to **Additional Library Directories**, type in the path:
        C:\libssh2-1.4.3\win32\Debug_lib;c:\openssl_lib\lib

Since the OpenSSL library was already included as part of the libSSH2 compilation, at the end of the libcurl compilation, the linker will present a whole bunch of “second definition ignored” warnings, which can be safely ignored.