Building OpenSSL with Visual Studio

New:

- Now supports both the 1.0.1 and 1.0.2 branch
- Now supports Visual Studio 2010, 2013 and 2015
- Now supports both static and dynamic libraries

Downloads

I provide downloads for Visual Studio 2010 and 2015. I had to remove my 2013 installation due to space constraints, but the build files are still there so you can do it, too.

Version	Visual Studio 2010	Visual Studio 2015

OpenSSL 1.0.2d	32-Bit Release DLL	32-Bit Release DLL
	32-Bit Debug DLL	32-Bit Debug DLL
	32-Bit Release Static Library	32-Bit Release Static Library
	32-Bit Debug Static Library	32-Bit Debug Static Library
	64-Bit Release DLL	64-Bit Release DLL
	64-Bit Debug DLL	64-Bit Debug DLL
	64-Bit Release Static Library	64-Bit Release Static Library
	64-Bit Debug Static Library	64-Bit Debug Static Library
OpenSSL 1.0.1p	32-Bit Release DLL	32-Bit Release DLL
	32-Bit Debug DLL	32-Bit Debug DLL
	32-Bit Release Static Library	32-Bit Release Static Library [broken]
	32-Bit Debug Static Library	32-Bit Debug Static Library [broken]
	64-Bit Release DLL	64-Bit Release DLL
	64-Bit Debug DLL	64-Bit Debug DLL
	64-Bit Release Static Library	64-Bit Release Static Library [broken]
	64-Bit Debug Static Library	64-Bit Debug Static Library [broken]

Building OpenSSL automatically

Because the process of building OpenSSL is time consuming and error prone, I wrote a couple of batch scripts that simplify the process significantly. You can download them here. Inside, you will find three batch files:

- `rebuild_openssl_vs2010.cmd` for use with Visual Studio 2010
- `rebuild_openssl_vs2013.cmd` for use with Visual Studio 2013
- `rebuild_openssl_vs2015.cmd` for use with Visual Studio 2015

Prerequisites

- The script assumes you are on Windows.
- The script assumes you have Visual Studio 2010, 2013 or 2015 installed in all the usual places. Important: If you have a different installation folder, your mileage may vary
- The script assumes you have downloaded an OpenSSL tarball, like this one.
- The script assumes you have Python (2.7 or 3.x) installed and on your PATH
- The script assumes you have 7-zip installed (doesn't need to be on your PATH)
- Choose the script you want to use and edit it. For example, let's take a look at the top of `rebuild_openssl_vs2015.cmd`:

```
T:
set OPENSSL_VERSION=1.0.1p
set SEVENZIP="C:\Program Files\7-Zip\7z.exe"
set VS2015="C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\vcvars32.bat"
set VS2015_AMD64="C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\amd64\vcvars64.bat"
```

so it is pretty easy to see: you must enter the OpenSSL version manually, the rest should have

sensible defaults...

- **Note:** The script uses the `SUBST` T:\ drive for building OpenSSL.

Building the OpenSSL binaries

- Place the tar.gz file (not the unpacked .tar) in the root of T:\
- Double-click on one of the `rebuild_openssl-vs*.cmd`.

That's it, it will do all the hard work for you and present nicely packaged binaries. Great fun!

Building OpenSSL manually

OK, so you don't trust me. Right. Well, here is how you can do it manually.... **Note:** This article wouldn't have been possible without the invaluable help of this article. However, it was obviously not built on a Windows 8 machine, and it didn't include any binaries. So this article follows the same basic structure, but it has some important differences:

- The instructions default to *the DLL build* Why? because that is the one used by Python. And because I was rebuilding Python, I was rebuilding OpenSSL in the first place. So there.
- Debug build uses .PDBs

Prerequisites

- You need Visual Studio 2010, 2013 or 2015.
- You need to install Perl. I used ActivePerl 5.16.3 for Windows (x86).
- You need the OpenSSL sourcecode. In the following, both the 1.0.1 and 1.0.2 branches are supported.
- Unzip the sourcecode.
- Create *two different copies of the sourcecode* I am going to follow the conventions of the original article and create `T:\openssl-src-32` and `T:\openssl-src-64`.
- You need a development prompt. This varies based on your compiler:
  - **Visual Studio 2010, 32-bit:** Open `Visual Studio Command Prompt (2010)`
  - **Visual Studio 2010, 64-bit:** Open `Visual Studio x64 Win64 Command Prompt (2010)`
  - **Visual Studio 2013, 32-bit:** Open `CMD.EXE` and run `C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\vcvars32.bat`
  - **Visual Studio 2013, 64-bit:** Open `CMD.EXE` and run `C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\bin\amd64\vcvars64.bat`
  - **Visual Studio 2015, 32-bit:** Open `CMD.EXE` and run `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\vcvars32.bat`
  - **Visual Studio 2015, 64-bit:** Open `CMD.EXE` and run `C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\amd64\vcvars64.bat`

Building the 32-bit Release DLL

- Change to the source directory, for example `T:\openssl-src-32`

- Run `perl Configure VC-WIN32 --prefix=T:\Build-OpenSSL-VC32-Release-DLL`. This will make `T:\Build-OpenSSL-VC32-Release-DLL` your output directory; it should be fairly obvious how you can change that.
- Run `ms\do_ms`
- **If you are on the 1.0.2 branch, you must do the following:** Run `ms\do_nasm`.
  This step is **not** necessary if you are on 1.0.1
- Run `nmake -f ms\ntdll.mak`
- Run `nmake -f ms\ntdll.mak install`

Building the 32-bit Debug DLL

- Change to the source directory, for example `T:\openssl-src-32`
- Run `perl Configure debug-VC-WIN32 --prefix=T:\Build-OpenSSL-VC32-Debug-DLL`
- Run `ms\do_ms`
- **If you are on the 1.0.2 branch, you must do the following:** Run `ms\do_nasm`.
  This step is **not** necessary if you are on 1.0.1
- Run `nmake -f ms\ntdll.mak`
- Run `nmake -f ms\ntdll.mak install`

Building the 32-bit Release Static Library

- Change to the source directory, for example `T:\openssl-src-32`
- Run `perl Configure VC-WIN32 --prefix=T:\Build-OpenSSL-VC32-Release-DLL`. This will make `T:\Build-OpenSSL-VC32-Release-DLL` your output directory; it should be fairly obvious how you can change that.
- Run `ms\do_ms`
- **If you are on the 1.0.2 branch, you must do the following:** Run `ms\do_nasm`. This step is **not** necessary if you are on 1.0.1
- Run `nmake -f ms\nt.mak`
- Run `nmake -f ms\nt.mak install`

Building the 32-bit Debug Static Library

- Change to the source directory, for example `T:\openssl-src-32`
- Run `perl Configure debug-VC-WIN32 --prefix=T:\Build-OpenSSL-VC32-Debug-DLL`
- Run `ms\do_ms`
- **If you are on the 1.0.2 branch, you must do the following:** Run `ms\do_nasm`. This step is **not** necessary if you are on 1.0.1
- Run `nmake -f ms\nt.mak`
- Run `nmake -f ms\nt.mak install`

Building the 64-bit Release DLL

- Change to the source directory, for example `T:\openssl-src-64`
- Run `perl Configure VC-WIN64A --prefix=T:\Build-OpenSSL-VC64-Release-DLL`. This will make `T:\Build-OpenSSL-VC64-Release-DLL` your output directory; it should be fairly obvious how you can change that.
- Run `ms\do_win64a`
- Run `nmake -f ms\ntdll.mak`
- Run `nmake -f ms\ntdll.mak install`

Building the 64-bit Debug DLL

- Change to the source directory, for example `T:\openssl-src-64`
- Run `perl Configure debug-VC-WIN64A --prefix=T:\Build-OpenSSL-VC64-Debug-DLL`
- Run `ms\do_win64a`
- Run `nmake -f ms\ntdll.mak`
- Run `nmake -f ms\ntdll.mak install`

Building the 64-bit Release Static Library

- Change to the source directory, for example `T:\openssl-src-32`
- Run `perl Configure VC-WIN64A --prefix=T:\Build-OpenSSL-VC64-Release-Static`. This will make `T:\Build-OpenSSL-VC64-Release-Static` your output directory; it should be fairly obvious how you can change that.
- Run `ms\do_win64a`
- Run `nmake -f ms\nt.mak`
- Run `nmake -f ms\nt.mak install`

Building the 64-bit Debug Static Library

- Change to the source directory, for example `T:\openssl-src-64`
- Run `perl Configure debug-VC-WIN64A --prefix=T:\Build-OpenSSL-VC64-Debug-DLL`
- Run `ms\do_win64a`
- Run `nmake -f ms\nt.mak`
- Run `nmake -f ms\nt.mak install`

FAQ

Why did I need to create two different copies of the sourcecode

Because the OpenSSL build scripts will use the folde`outdll32` for *both the 32-bit and the 64-bit output*, so there is no easy way to distinguish both builds.

GK, December 12, 2015

Note: Special thanks toAlex (see https://github.com/CpServiceSpb/OpenSSLOcsp) for pointing out some mistakes in the documentation of the 64-bit build. Should be fine now.