How to build third party libraries with Visual Studio 2010/2012/2013

The Vertexwahn Build Guide

Content

[BlueGo 2](#_Toc435703955)

[Boost 2](#_Toc435703956)

[Boost 1.50.0 with Visual Studio 2010 x64 2](#_Toc435703957)

[Boost 1.52.0 with Visual Studio 2010 x64 3](#_Toc435703958)

[Boost 1.51.0 with Visual Studio 2012 x64 4](#_Toc435703959)

[Boost 1.53.0 with Visual Studio 2012 x64 4](#_Toc435703960)

[Boost 1.54.0 with Visual Studio 2012 x64 4](#_Toc435703961)

[Boost 1.55.0 with Visual Studio 2012 x64 5](#_Toc435703962)

[Boost 1.57.0 with Visual Studio 2013 x64 5](#_Toc435703963)

[Boost from source with Visual Studio 2013 x64 5](#_Toc435703964)

[Boost 1.59.0 with Visual Studio 2015 x64 5](#_Toc435703965)

[Qt 6](#_Toc435703966)

[Qt 4.8.2 with Visual Studio 2010 x64 6](#_Toc435703967)

[Qt 4.8.3 with Visual Studio 2010 x64 6](#_Toc435703968)

[Qt 5.0.2 with Visual Studio 2010 x64 6](#_Toc435703969)

[Qt 4.8.3 with Visual Studio 2012 x64 7](#_Toc435703970)

[Qt 5.0.0 Beta1 with Visual Studio 2012 x64 7](#_Toc435703971)

[Qt 5.0.0 with Visual Studio 2012 x64 7](#_Toc435703972)

[How to use it with CMake 8](#_Toc435703973)

[Parallel Builds 9](#_Toc435703974)

[Qt 5.0.2 with Visual Studio 2012 x64 9](#_Toc435703975)

[Qt 5.1.0 with Visual Studio 2012 x64 9](#_Toc435703976)

[Qt 5.1.1 with Visual Studio 2013 x64 9](#_Toc435703977)

[OpenSceneGraph 10](#_Toc435703978)

[How to build OpenSceneGraph 3.0.1 with Visual Studio 2012 x64? 10](#_Toc435703979)

[libLAS 11](#_Toc435703980)

[Build libLAS-1.7.0 with Visual Studio 2013 x64 11](#_Toc435703981)

[Build libLAS-1.7.0 with Visual Studio 2012 x64 13](#_Toc435703982)

[Build libLAS-1.7.0 with Visual Studio 2010 x64 14](#_Toc435703983)

[wxWidgets 16](#_Toc435703984)

[How to build wxWidgets 2.9.3 with Visual Studo 2010 x64? 16](#_Toc435703985)

# BlueGo

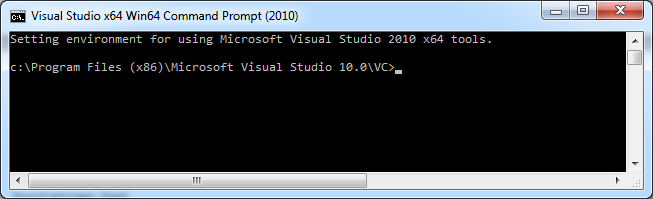
BlueGo is a third party library build tool that helps you to build boost, qt and OpenSceneGraph libraries. It can be downloaded from vertexwahn.de. But you can also build the libraries manually as described in this guide.

# Boost

## Boost 1.50.0 with Visual Studio 2010 x64

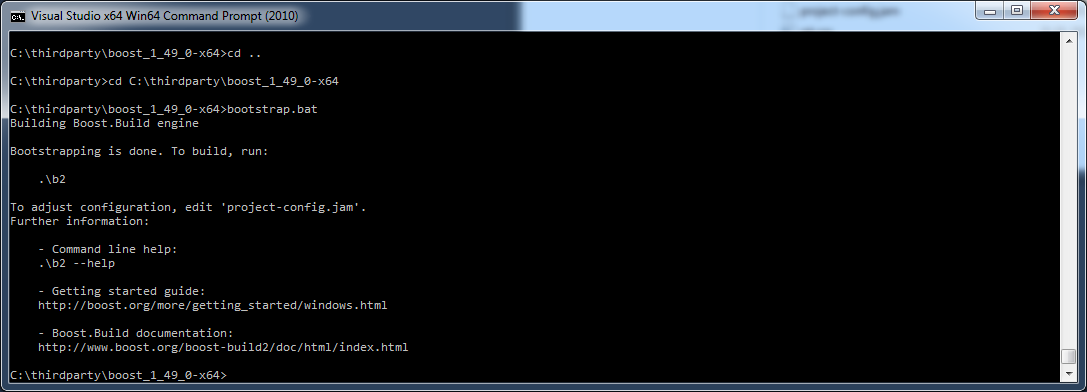
In this section the build process for a x64 build target is described.

1. Download boost 1.50.0 from <http://www.boost.org/>
2. Extract files (e.g. “C:\thirdpatry\boost\_1\_50\_0-x64”)
3. start Visual Studio 2010 x64 command prompt (“Visual Studio x64 Win64 Command Prompt (2010)”)



1. Change to boost directory (e.g. “cd C:\thirdpatry\boost\_1\_50\_0-x64”, this is the directory which contains a bootstrap.bat)
2. Execute

.\bootstrap.bat



And

.\b2 address-model=64 --build-type=complete stage

You can also build boost parallel. Just use the command line argument -j.

.\b2 –j8 address-model=64 --build-type=complete stage

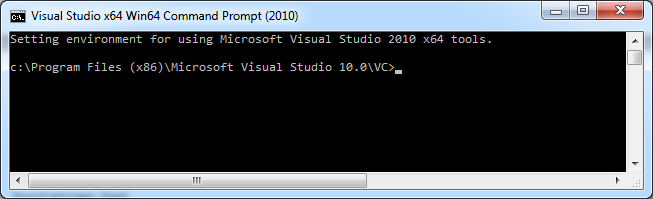
-j8 means that 8 threads are started to build boost.

Successfully tested with Visual Studio 10 x64

## Boost 1.52.0 with Visual Studio 2010 x64

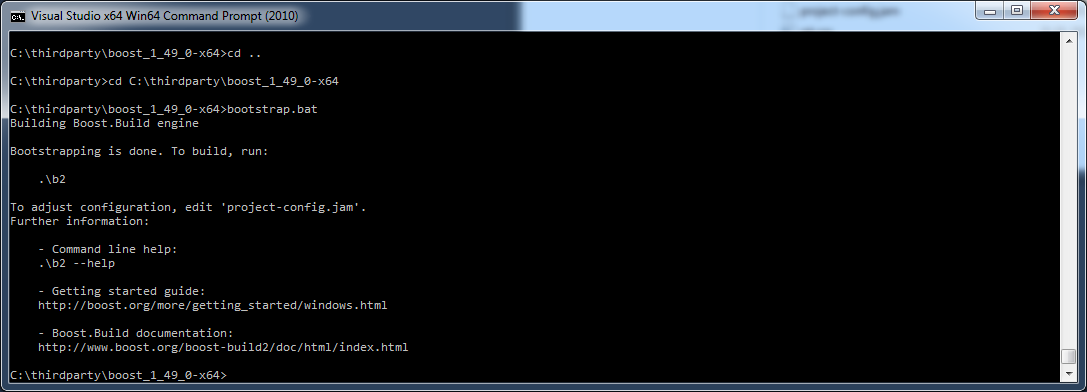
In this section the build process for a x64 build target is described.

1. Download boost 1.52.0 from <http://www.boost.org/>
2. Extract files (e.g. “C:\thirdparty\vs2010\x64\boost\_1\_52\_0”)
3. Start Visual Studio 2010 x64 command prompt (“Visual Studio x64 Win64 Command Prompt (2010)”)



1. Change to boost directory (e.g. “cd C:\thirdparty\vs2010\x64\boost\_1\_52\_0”)
2. Go to the directory tools\build\v2
3. Execute

.\bootstrap.bat



1. Copy “b2.exe” to the path where bootstrap.bat resides.
2. Execute

b2 –j4 address-model=64 **toolset=**[*msvc*](http://www.boost.org/doc/libs/1_52_0/more/getting_started/windows.html#toolset-name)-10.0 **--build-type=complete** stage

## Boost 1.51.0 with Visual Studio 2012 x64

1. change directory to boost\_1\_51\_0-x64\tools\build\v2
2. execute bootstrap.bat (not from the the native x64 comand prompt - just double click it)
3. copy b2.exe back to boost\_1\_51\_0-x64 dir
4. Open VS2012 x64 Native Tools command prompt
5. change directory to boost\_1\_51\_0-x64
6. execute b2

## Boost 1.53.0 with Visual Studio 2012 x64

1. Download boost 1.53.0 from <http://www.boost.org/>
2. Extract files (e.g. “C:\thirdparty\vs2010\x64\boost\_1\_53\_0”)
3. Start Visual Studio 2012 x64 command prompt (“Open VS2012 x64 Native Tools Command Prompt“)
4. Change to boost directory (e.g. “cd C:\thirdparty\vs2012\x64\boost\_1\_53\_0”)
5. Go to the directory tools\build\v2
6. Execute .\bootstrap.bat
7. Copy “b2.exe” to the path where bootstrap.bat resides.
8. Execute b2 –j4 address-model=64 toolset=msvc-11.0 --build-type=complete stage

b2 –j4 address-model=64 toolset=msvc-11.0 --build-type=complete stage --with-filesystem --with-signals --with-system

## Boost 1.54.0 with Visual Studio 2012 x64

1. Download boost 1.54.0 from <http://www.boost.org/>
2. Extract files (e.g. “C:\thirdparty\vs2010\x64\boost\_1\_54\_0”)
3. Start Visual Studio 2012 x64 command prompt (“Open VS2012 x64 Native Tools Command Prompt“)
4. Change to boost directory (e.g. “cd C:\thirdparty\vs2012\x64\boost\_1\_54\_0”)
5. Execute .\bootstrap.bat
6. Execute b2 address-model=64 toolset=msvc-11.0 --build-type=complete stage

If only certain libraries (for instance filesystem) are need step 6 can be replaced by this:

b2 address-model=64 toolset=msvc-11.0 --build-type=complete stage --with-filesystem --with-signals --with-system

## Boost 1.55.0 with Visual Studio 2012 x64

1. Download boost 1.55.0 from <http://www.boost.org/>
2. Extract files (e.g. “C:\thirdparty\vs2012\x64\boost\_1\_55\_0”)
3. Start Visual Studio 2012 x64 command prompt (“Open VS2012 x64 Native Tools Command Prompt“)
4. Change to boost directory (e.g. “cd C:\thirdparty\vs2012\x64\boost\_1\_55\_0”)
5. Execute .\bootstrap.bat
6. Execute b2 address-model=64 toolset=msvc-11.0 --build-type=complete stage

If only certain libraries (for instance filesystem) are need step 6 can be replaced by this:

b2 address-model=64 toolset=msvc-11.0 --build-type=complete stage --with-filesystem --with-signals --with-system

## Boost 1.57.0 with Visual Studio 2013 x64

1. Download boost 1.57.0 from <http://www.boost.org/>
2. Extract files (e.g. “C:\thirdparty\vs2013\x64\boost\_1\_57\_0”)
3. Start Visual Studio 2013 x64 command prompt (“VS2013 x64 Native Tools Command Prompt “)
4. Change to boost directory (e.g. “cd C:\thirdparty\vs2013\x64\boost\_1\_57\_0”)
5. Execute .\bootstrap.bat
6. Execute b2 -j8 --toolset=msvc-12.0 address-model=64 --build-type=complete stage
7. b2 -j8 --toolset=msvc-12.0 address-model=64 --build-type=complete stage --with-python

## Boost from source with Visual Studio 2013 x64

1. Checkout the source via git clone --recursive https://github.com/boostorg/boost.git boost\_1\_56\_0\_alpha
2. cd boost\_1\_56\_0\_alpha
3. .\bootstrap
4. .\b2 headers
5. b2 address-model=64 toolset=msvc-12.0 --build-type=complete stage

Checkout the source via git clone --recursive https://github.com/boostorg/boost.git boost\_1\_56\_0\_alpha

cd boost\_1\_56\_0\_alpha

.\bootstrap

.\b2 headers

b2 address-model=64 toolset=msvc-12.0 --build-type=complete stage

## Boost 1.59.0 with Visual Studio 2015 x64

1. Download boost 1.59.0 from <http://www.boost.org/>
2. Extract files (e.g. “C:\thirdparty\vs2013\x64\boost\_1\_59\_0”)
3. Start Visual Studio 2015 x64 command prompt (“VS2015 x64 Native Tools Command Prompt “)
4. Change to boost directory (e.g. “cd C:\thirdparty\vs2013\x64\boost\_1\_59\_0”)
5. Execute .\bootstrap.bat
6. Execute b2 -j8 --toolset=msvc-14.0 address-model=64 --build-type=complete stage

# Qt

A good descripton of how to Compile Qt 4.7 with Visual Studio 2010 can be found at: <http://www.holoborodko.com/pavel/2011/02/01/how-to-compile-qt-4-7-with-visual-studio-2010/>

## Qt 4.8.2 with Visual Studio 2010 x64

1. Ensure that Service Pack 1 is installed for Visual Studio 2010
2. Install strawbey perl (<http://strawberryperl.com/>)
3. Download Qt 4.8.2 source code (http://download.qt.nokia.com/qt/source/qt-everywhere-opensource-src-4.8.2.zip) from <http://qt.nokia.com/downloads>
4. Start “Visual Studio x64 Win64 Command Prompt (2010)“
5. Switch to Qt 4.8.2 source directory (directory which contains configure.exe)
6. Run

configure -mp -opensource -nomake demos -nomake examples -platform win32-msvc2010

1. and afterwards

nmake

Approved through testing on Acer Notebook

## Qt 4.8.3 with Visual Studio 2010 x64

1. Ensure that Service Pack 1 is installed for Visual Studio 2010
2. Install strawbey perl (<http://strawberryperl.com/>)
3. Download Qt 4.8.3 source code from <http://qt-project.org/downloads>
4. Start “Visual Studio x64 Win64 Command Prompt (2010)“
5. Switch to Qt 4.8.3 source directory (directory which contains configure.exe)
6. Run

configure -mp -opensource -nomake demos -nomake examples -platform win32-msvc2010

1. and afterwards

nmake

## Qt 5.0.2 with Visual Studio 2010 x64

1. Install strawbey perl (<http://strawberryperl.com/>)
2. Install Windows SDK http://msdn.microsoft.com/en-us/windows/desktop/aa904949 even when you are using Windows 7. On older versions of Microsoft Windows the DirectX SDK should be installed.
3. Download Qt 5.0.2 source code from <http://qt-project.org/downloads>
4. Start “Visual Studio x64 Win64 Command Prompt (2010)”
5. Make sure that python is available on the command prompt. When installing Python you can add python.exe to the environment path. (I used python 3.3)
6. Switch to Qt 5.0.2 source directory (directory which contains configure.bat)
7. Run configure -developer-build -opensource -nomake examples -nomake tests -opengl
8. and afterwards

nmake

## Qt 4.8.3 with Visual Studio 2012 x64

1. Install strawbey perl (<http://strawberryperl.com/>)
2. Download Qt 4.8.3 source code from <http://qt-project.org/downloads>
3. Go to mkspecs\win32-msvc2010.

Open qmake.conf and change

QMAKE\_COMPILER\_DEFINES += \_MSC\_VER=1600 WIN32

to

QMAKE\_COMPILER\_DEFINES += \_MSC\_VER=1700 WIN32

1. Replace qt-everywhere-opensource-src-4.8.3-x64\src\3rdparty\webkit\Source\JavaScriptCore\wtf\HashSet.h by provided HashSet.h
2. Start “VS2012 x64 Native Tools Command Prompt“
3. Switch to Qt 4.8.3 source directory (directory which contains configure.exe)
4. Run

configure -mp -opensource -nomake demos -nomake examples -platform win32-msvc2010

1. and afterwards

nmake

Successfully tested on Dev0.

References:  
<http://stackoverflow.com/questions/12113400/compiling-qt-4-8-2-for-visual-studio-2012>  
<http://qt-project.org/forums/viewthread/17771>

## Qt 5.0.0 Beta1 with Visual Studio 2012 x64

1. Install strawberryperl (<http://strawberryperl.com/>)
2. Start “Open VS2012 x64 Native Tools Command Prompt“
3. Run

configure -mp -opensource -nomake demos -nomake examples -platform win32-msvc2012

1. and afterwards

nmake

* does not work

What I did, I went to win32-msvc2010.

There, open qmake.conf and change

QMAKE\_COMPILER\_DEFINES += \_MSC\_VER=1600 WIN32

to

QMAKE\_COMPILER\_DEFINES += \_MSC\_VER=1700 WIN32

Then I could configure using win32-msvc2010.

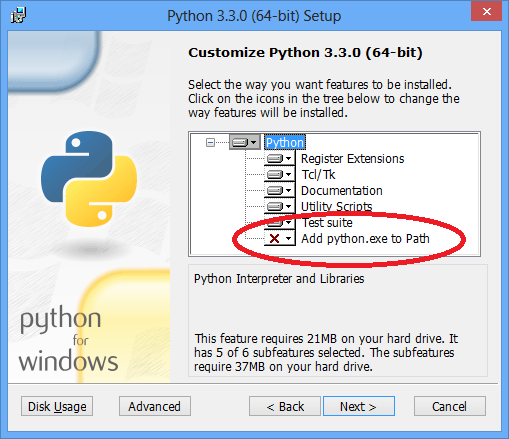
Also, I used these changes: <http://qt-project.org/forums/viewthread/17771> (alzi's Version)

open

C:\thirdparty\vs2012\x64\qt-everywhere-opensource-src-4.8.2\src\3rdparty\webkit\Source\JavaScriptCore\wtf\HashSet.h

## Qt 5.0.0 with Visual Studio 2012 x64

1. Install strawbey perl (<http://strawberryperl.com/>)
2. Install Windows SDK http://msdn.microsoft.com/en-us/windows/desktop/aa904949 even when you are using Windows 7. On older versions of Microsoft Windows the DirectX SDK should be installed.
3. Download Qt 5.0.0 source code from <http://qt-project.org/downloads>
4. Start “VS2012 x64 Native Tools Command Prompt“
5. Make sure that python is available on the command prompt. When installing Python you can add python.exe to the environment path. (I used python 3.3)



1. Switch to Qt 5.0.0 source directory (directory which contains configure.exe)
2. Run configure -developer-build -opensource -nomake examples -nomake tests
3. and afterwards

nmake

### How to use it with CMake

Afterwards set Qt5Core\_DIR to “C:/thirdparty/vs2012/x64/qt-everywhere-opensource-src-5.0.1/qtbase/lib/cmake/Qt5Core”

It might be easier to just set the environment variable CMAKE\_PREFIX\_PATH to the cmake directory, as explained there: <http://doc-snapshot.qt-project.org/5.0/qtdoc/cmake-manual.html>

"In order for find\_package to be successful, Qt 5 must be found below the CMAKE\_PREFIX\_PATH, or the Qt5<Module>\_DIR must be set in the CMake cache to the location of the Qt5WidgetsConfig.cmake file. The easiest way to use CMake is to set the CMAKE\_PREFIX\_PATH environment variable to the install prefix of Qt 5."

References: Building Qt 5 from Git <http://qt-project.org/wiki/Building_Qt_5_from_Git>

### Parallel Builds

* Pass the option *-MP* to Qt’s *configure*
* Set the environment variable *CL* (specifying Visual Studio compiler options) to */MP* (On the command line: set CL=/MP)

## Qt 5.0.2 with Visual Studio 2012 x64

1. Install strawbey perl (<http://strawberryperl.com/>)
2. Install Windows SDK http://msdn.microsoft.com/en-us/windows/desktop/aa904949 even when you are using Windows 7. On older versions of Microsoft Windows the DirectX SDK should be installed.
3. Download Qt 5.0.2 source code from <http://qt-project.org/downloads>
4. Start “Open VS2012 x64 Native Tools Command Prompt“
5. Make sure that python is available on the command prompt. When installing Python you can add python.exe to the environment path. (I used python 3.3)
6. Switch to Qt 5.0.2 source directory (directory which contains configure.bat)
7. Run configure -developer-build -opensource -nomake examples -nomake tests
8. and afterwards

nmake

## Qt 5.1.0 with Visual Studio 2012 x64

1. Install strawbey perl (<http://strawberryperl.com/>)
2. Install Windows SDK http://msdn.microsoft.com/en-us/windows/desktop/aa904949 even when you are using Windows 7. On older versions of Microsoft Windows the DirectX SDK should be installed.
3. Download Qt 5.1.0 source code from <http://qt-project.org/downloads>
4. Start “Open VS2012 x64 Native Tools Command Prompt“
5. Make sure that python is available on the command prompt. When installing Python you can add python.exe to the environment path. (I used python 3.3)
6. Switch to Qt 5.1.0 source directory (directory which contains configure.bat)
7. Run configure -developer-build -opensource -nomake examples -nomake tests
8. and afterwards

nmake

## Qt 5.1.1 with Visual Studio 2013 x64

1. Install AcrivePerl from <http://www.activestate.com/activeperl>
2. Install python <http://www.python.org/download/releases/3.3.3/>
3. Get Ruby from: [http://www.rubyinstaller.org/downloads/](http://www.rubyinstaller.org/)
4. Install Windows SDK 8.1 from <http://msdn.microsoft.com/en-us/windows/desktop/aa904949> even when you are using Windows 7. On older versions of Microsoft Windows the DirectX SDK should be installed.
5. Download Qt 5.1.1 source code: <http://download.qt-project.org/official_releases/qt/5.1/5.1.1/single/qt-everywhere-opensource-src-5.1.1.zip>
6. Start “VS2013 x64 Native Tools Command Prompt“
7. configure -developer-build -opensource -nomake examples -nomake tests
8. nmake
9. Execute git checkout stable<http://qt-project.org/wiki/Building_Qt_5_from_Git>
10. Make sure that python is available on the command prompt. When installing Python you can add python.exe to the environment path. (I used python 3.3)
11. Switch to Qt 5.1.1 source directory (directory which contains configure.bat)

C:/thirdparty/vs2012/x64/qt-everywhere-opensource-src-5.1.1

1. Run configure -developer-build -opensource -nomake examples -nomake tests
2. and afterwards

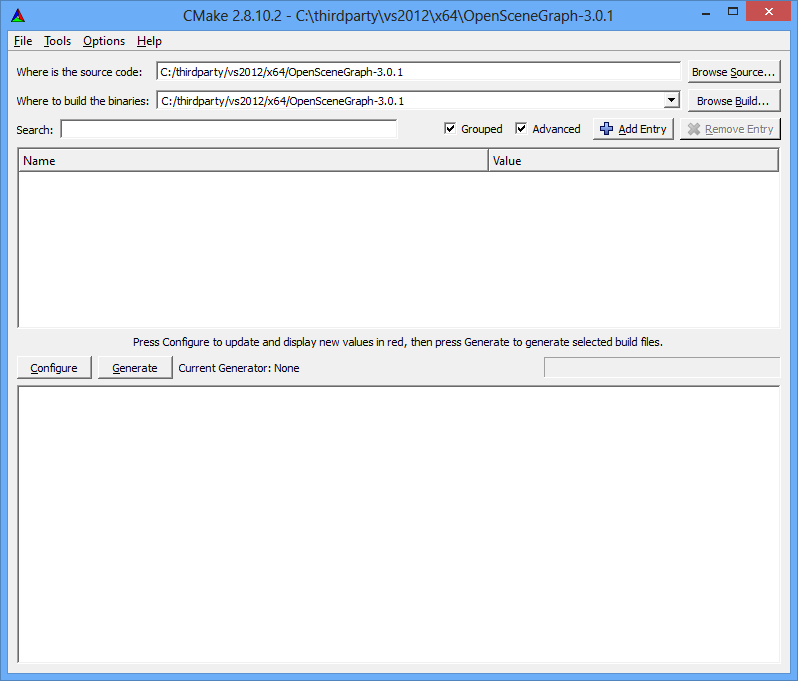
nmake

# OpenSceneGraph

## How to build OpenSceneGraph 3.0.1 with Visual Studio 2012 x64?

This post should show how OpenSceneGraph 3.0.1 can be built using Visual Studio 2012 x64.

1. Fist you need to download the sourcecode from [OpenSceneGraph-3.0.1.zip](http://www.openscenegraph.org/downloads/stable_releases/OpenSceneGraph-3.0.1/source/OpenSceneGraph-3.0.1.zip)
2. Extract folder (for instance to "C:\thirdparty\vs2012\x64\OpenSceneGraph-3.0.1")
3. To enable a parallel build procecess add the following lines to the top most CMakeLists.txt file:
   1. # Compiler settings
   2. **if** (MSVC11)
   3. # Standard: /DWIN32 /D\_WINDOWS /W3 /Zm1000 /EHsc /GR
   4. **set**(CMAKE\_CXX\_FLAGS "/DWIN32 /D\_WINDOWS /W4 /Zi /EHsc /GR- /MP /openmp")
   5. endif (MSVC11)
4. Now generate a Visual Studio 2012 x64 Solution via CMake

[](file:///C:\dev\devertexwahn\cmakeOSG.png)

5. Open the generated solution file.

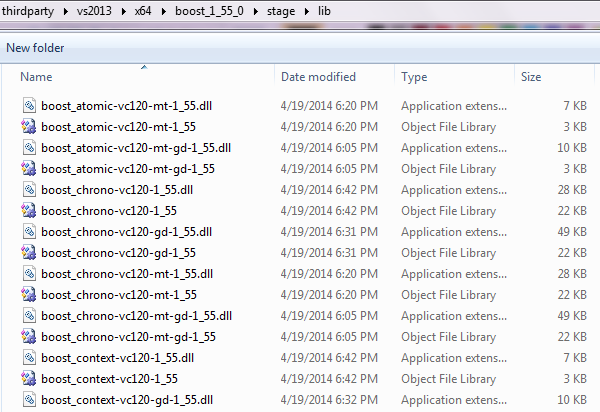
1. Build for example the project "Applications osgversion" in debug and release mode. This will generated the necessary lib and dll files.

# libLAS

## Build libLAS-1.7.0 with Visual Studio 2013 x64

The following steps list out how to build libLAS-1.7.0 using Visual Studio 2013 x64.

1. Download the source code from [libLAS-1.7.0.zip](http://download.osgeo.org/liblas/libLAS-1.7.0.zip)
2. Extract libLAS-1.7.0.zip (for instance to "C:\thirdparty\vs2013\x64\libLAS-1.7.0")
3. To build libLAS successfully, the header files and 64 bit binaries for Boostmust be accessible (for instance to "C:\thirdparty\vs2013\x64\boost\_1\_55\_0")
4. Ensure that stage\lib (for instance to "C:\thirdparty\vs2013\x64\boost\_1\_55\_0\stage\bin") folder contains Boost’s libraries. Snapshot below gives an idea of the folder structure of stage\lib.



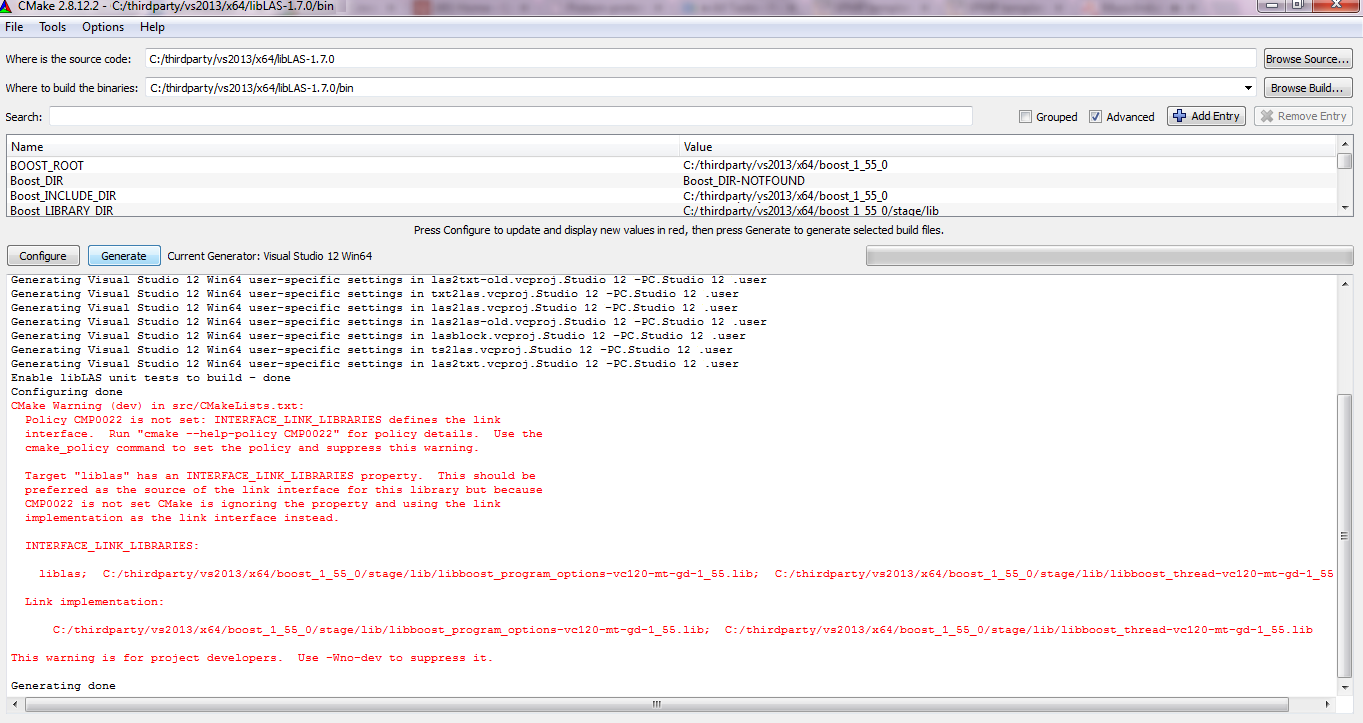
1. Create a folder to generate a Visual Studio 2012 x64 solution via CMake (for instance bin folder: "C:\thirdparty\vs2013\x64\libLAS-1.7.0\bin")
2. This step will generate build files under folder created in step 5. Execute CMake command from the windows command prompt with following parameters (for detailed parameter description refer to CMake usages):

"C:\Program Files (x86)\CMake 2.8\bin\cmake.exe" -G"Visual Studio 12 Win64"

-H"C:\thirdparty\vs2013\x64\libLAS-1.7.0" -B"C:\thirdparty\vs2013\x64\libLAS-1.7.0\bin"

-DBOOST\_ROOT="C:\thirdparty\VS2013\x64\boost\_1\_55\_0"

or generate build files using CMake gui as shown in the snapshot below.

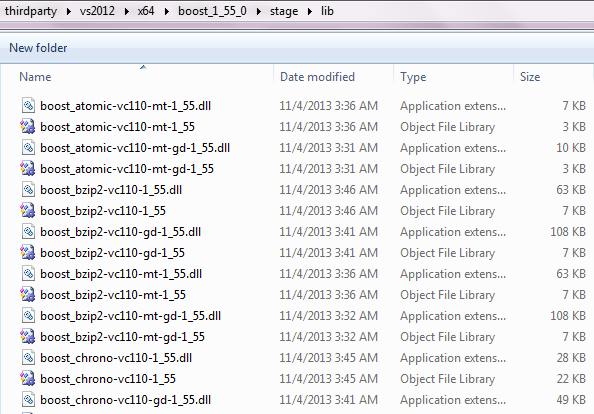


1. Open generated solution-libLAS.sln under bin folder (for instance : "C:\thirdparty\vs2013\x64\libLAS-1.7.0\bin")
2. Build the libLAS.sln to generate the necessary libs and dlls.
3. As an alternative to step 7,8: To build with command prompt, make sure that the latest MSBuild.exe is installed (for instance under "C:\Program Files (x86)\MSBuild\12.0\Bin\MSBuild.exe") and open windows command prompt to execute following commands:
   1. cd /D "C:\thirdparty\vs2013\x64\libLAS-1.7.0\bin"
   2. "C:\Program Files (x86)\MSBuild\12.0\Bin\MSBuild.exe" libLAS.sln /property:Configuration=Release
4. Verify in path (for instance: "C:\thirdparty\vs2013\x64\libLAS-1.7.0\bin\bin\Release") for the presence of necessary libs and dlls (for e.g. liblas.dll, liblas.lib etc.)

## Build libLAS-1.7.0 with Visual Studio 2012 x64

The following steps list out how to build libLAS-1.7.0 using Visual Studio 2012 x64.

1. Download the source code from [libLAS-1.7.0.zip](http://download.osgeo.org/liblas/libLAS-1.7.0.zip)
2. Extract libLAS-1.7.0.zip (for instance to "C:\thirdparty\vs2012\x64\libLAS-1.7.0")
3. To build libLAS successfully, the header files and 64 bit binaries for Boostmust be accessible (for instance to "C:\thirdparty\vs2012\x64\boost\_1\_55\_0")
4. Ensure that stage\lib (for instance to "C:\thirdparty\vs2012\x64\boost\_1\_55\_0\stage\bin") folder contains Boost’s libraries. Snapshot below gives an idea of the folder structure of stage\lib.



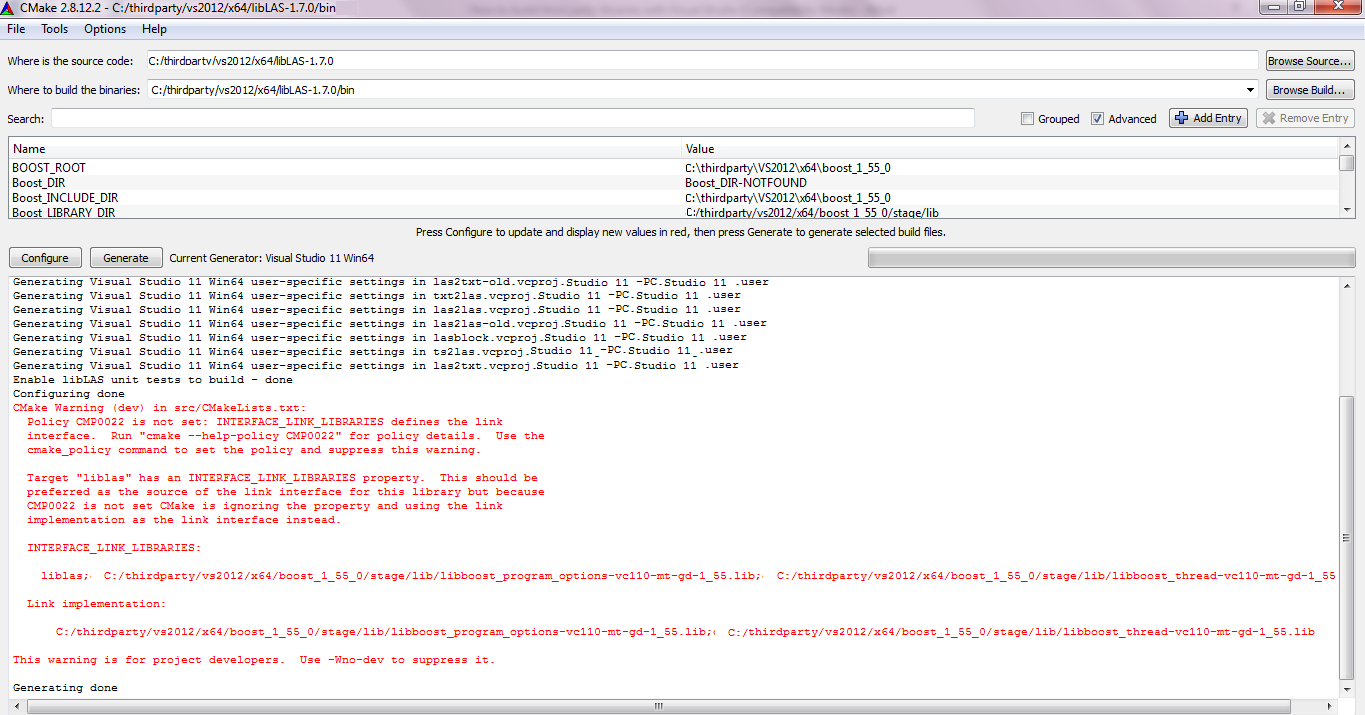
1. Create a folder to generate a Visual Studio 2011 x64 solution via CMake (for instance bin folder: "C:\thirdparty\vs2012\x64\libLAS-1.7.0\bin")
2. This step will generate build files under folder created in step 5. Execute CMake command from the windows command prompt with following parameters (for detailed parameter description refer to CMake usages):

"C:\Program Files (x86)\CMake 2.8\bin\cmake.exe" -G"Visual Studio 11 Win64"

-H"C:\thirdparty\vs2012\x64\libLAS-1.7.0" -B"C:\thirdparty\vs2012\x64\libLAS-1.7.0\bin"

-DBOOST\_ROOT="C:\thirdparty\VS2012\x64\boost\_1\_55\_0"

or generate build files using CMake gui as shown in the snapshot below.

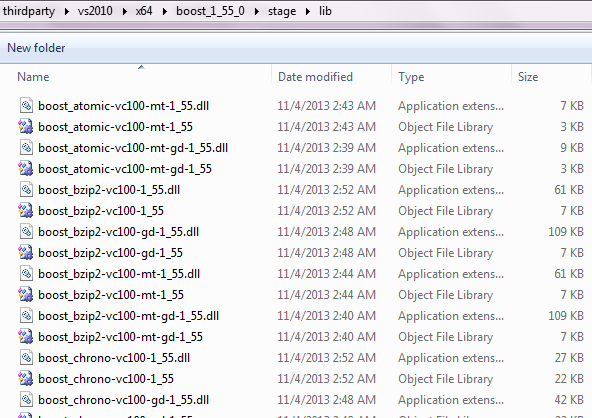


1. Open generated solution-libLAS.sln under bin folder (for instance : "C:\thirdparty\vs2012\x64\libLAS-1.7.0\bin")
2. Build the libLAS.sln to generate the necessary libs and dlls.
3. As an alternative to step 7,8: To build with command prompt, make sure that the latest MSBuild.exe is installed (for instance under "C:\Program Files (x86)\MSBuild\12.0\Bin\MSBuild.exe") and open windows command prompt to execute following commands:
   1. cd /D "C:\thirdparty\vs2012\x64\libLAS-1.7.0\bin"
   2. "C:\Program Files (x86)\MSBuild\12.0\Bin\MSBuild.exe" libLAS.sln /property:Configuration=Release
4. Verify in path (for instance: "C:\thirdparty\vs2012\x64\libLAS-1.7.0\bin\bin\Release") for the presence of necessary libs and dlls (for e.g. liblas.dll, liblas.lib etc.)

## Build libLAS-1.7.0 with Visual Studio 2010 x64

The following steps list out how to build libLAS-1.7.0 using Visual Studio 2010 x64.

1. Download the source code from [libLAS-1.7.0.zip](http://download.osgeo.org/liblas/libLAS-1.7.0.zip)
2. Extract libLAS-1.7.0.zip (for instance to "C:\thirdparty\vs2010\x64\libLAS-1.7.0")
3. To build libLAS successfully, the header files and 64 bit binaries for Boostmust be accessible (for instance to "C:\thirdparty\vs2010\x64\boost\_1\_55\_0")
4. Ensure that stage\lib (for instance to "C:\thirdparty\vs2010\x64\boost\_1\_55\_0\stage\bin") folder contains Boost’s libraries. Snapshot below gives an idea of the folder structure of stage\lib.



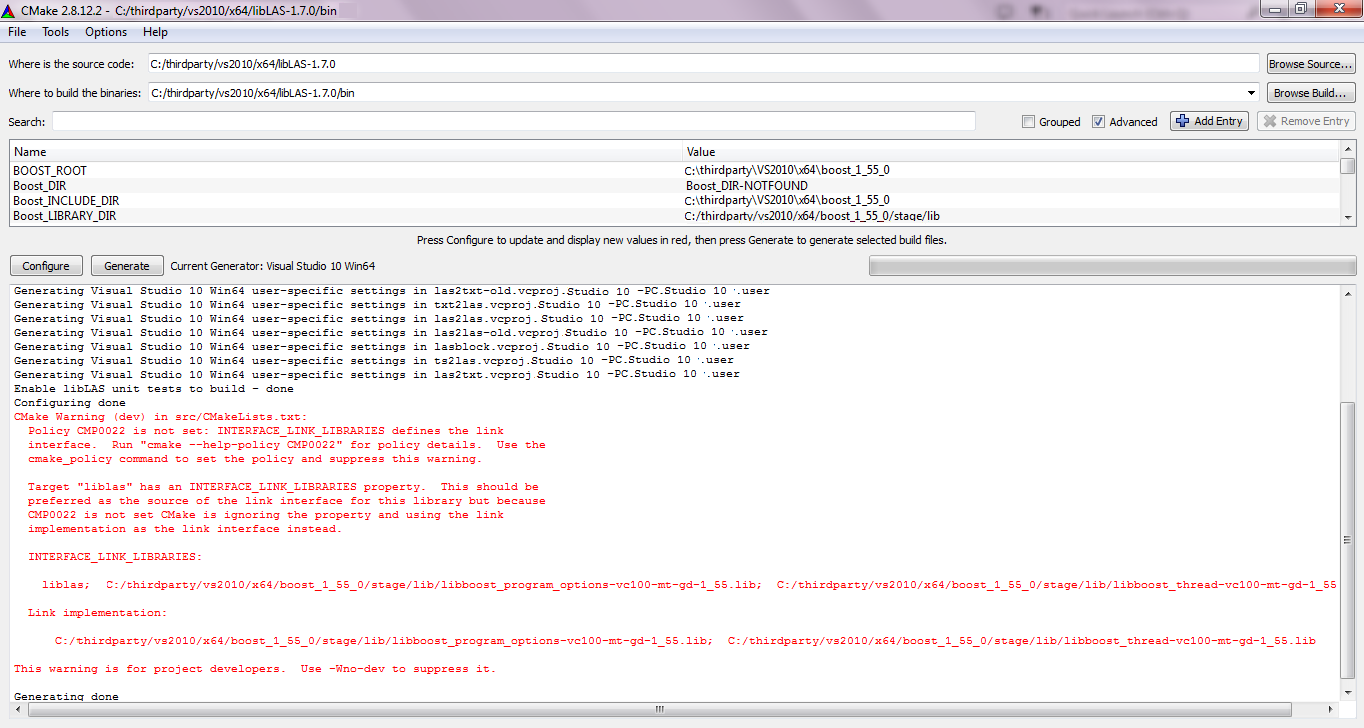
1. Create a folder to generate a Visual Studio 2010 x64 solution via CMake (for instance bin folder: "C:\thirdparty\vs2010\x64\libLAS-1.7.0\bin")
2. This step will generate build files under folder created in step 5. Execute CMake command from the windows command prompt with following parameters (for detailed parameter description refer to CMake usages):

"C:\Program Files (x86)\CMake 2.8\bin\cmake.exe" -G"Visual Studio 10 Win64"

-H"C:\thirdparty\vs2010\x64\libLAS-1.7.0" -B"C:\thirdparty\vs2010\x64\libLAS-1.7.0\bin"

-DBOOST\_ROOT="C:\thirdparty\VS2010\x64\boost\_1\_55\_0"

or generate build files using CMake gui as shown in the snapshot below.



1. Open generated solution-libLAS.sln under bin folder (for instance : "C:\thirdparty\vs2010\x64\libLAS-1.7.0\bin")
2. Build the libLAS.sln to generate the necessary libs and dlls.
3. As an alternative to step 7,8: To build with command prompt, make sure that the latest MSBuild.exe is installed (for instance under "C:\Program Files (x86)\MSBuild\12.0\Bin\MSBuild.exe") and open windows command prompt to execute following commands:
   1. cd /D "C:\thirdparty\vs2010\x64\libLAS-1.7.0\bin"
   2. "C:\Program Files (x86)\MSBuild\12.0\Bin\MSBuild.exe" libLAS.sln /property:Configuration=Release
4. Verify in path (for instance: "C:\thirdparty\vs2010\x64\libLAS-1.7.0\bin\bin\Release") for the presence of necessary libs and dlls (for e.g. liblas.dll, liblas.lib etc.)

# wxWidgets

## How to build wxWidgets 2.9.3 with Visual Studo 2010 x64?

1. Download wxWidgets-2.9.3.zip from <http://sourceforge.net/projects/wxwindows/files/2.9.3/>
2. Extract zip to a folder
3. Open Microsoft Visual Studio 2010 x64 tools command prompt
4. Change directory in VS command prompt to to the extracted folder (e.g. cd C:\thirdparty\wxWidgets-2.9.3-x64)
5. Build shared debug libraries: nmake -f makefile.vc UNICODE=1 BUILD="debug" SHARED=1
6. Build shared release libraries: nmake -f makefile.vc UNICODE=1 BUILD="release" SHARED=1

Now you can test if wxWiddgets works. Download the wxTestbed sample and use CMake to build it. You have only to set the wxWidget\_ROOT\_DIR to the path where you build wxWidgets (e.g. "C: hirdparty\wxWidgets-2.9.3-x64")