Google Git Sign in

chromium / external / github.com / google / protobuf / v3.0.0-beta-1 / . / **cmake**

tree: 68aa8b6bca807a90845146622c121e4ae2409bad [path history] [tgz]

- CMakeLists.txt
- README.md
- extract_includes.bat.in
- install.cmake
- libprotobuf-lite.cmake
- libprotobuf.cmake
- libprotoc.cmake
- protobuf-config-version.cmake.in
- protobuf-config.cmake.in
- protobuf-module.cmake.in
- protoc.cmake
- tests.cmake

cmake/README.md

This directory contains cmake files that can be used to generate MSVC project files in order to build protobuf on windows. You need to have cmake installed on your computer before proceeding.

Compiling and Installing

1. Check whether a gmock directory exists in the upper level directory. If you checkout the code from github via “git clone”, this gmock directory won't exist and you won't be able to build protobuf unit-tests. Consider using one of the release tar balls instead:

   ```
   https://github.com/google/protobuf/releases
   ```

   These release tar balls are more stable versions of protobuf and already have the gmock directory included.

   You can also download gmock by yourself and put it in the right place.

   If you absolutely don't want to build and run protobuf unit-tests, skip this step and use protobuf at your own risk.

2. Use cmake to generate MSVC project files. Running the following commands in a command shell will generate project files for Visual Studio 2008 in a sub-directory named “build”.

   ```
   $ cd path/to/protobuf/cmake
   $ mkdir build
   $ cd build
   $ cmake -G "Visual Studio 9 2008" ..
   ```

If you don't have gmock, skip the build of tests by turning off the BUILD_TESTING option:

```
$ cmake -G "Visual Studio 9 2008" -DBUILD_TESTING=OFF ..
```

3. Open the generated protobuf.sln file in Microsoft Visual Studio.
4. Choose “Debug” or “Release” configuration as desired.
5. From the Build menu, choose “Build Solution”. Wait for compiling to finish.
6. If you have built tests, run tests.exe and lite-test.exe from a command shell and check that all tests pass. Make sure you have changed the working directory to the output directory because tests.exe will try to find and run test_plugin.exe in the working directory.
7. Run extract_includes.bat to copy all the public headers into a separate “include” directory. This batch script can be found along with the generated protobuf.sln file in the same directory.
8. Copy the contents of the include directory to wherever you want to put headers.
9. Copy protoc.exe wherever you put build tools (probably somewhere in your PATH).
10. Copy libprotobuf.lib, libprotobuf-lite.lib, and libprotoc.lib wherever you put libraries.

To avoid conflicts between the MSVC debug and release runtime libraries, when compiling a debug build of your application, you may need to link against a debug build of libprotobuf.lib. Similarly, release builds should link against release libs.

DLLs vs. static linking

Static linking is now the default for the Protocol Buffer libraries. Due to issues with Win32‘s use of a separate heap for each DLL, as well as binary compatibility issues between different versions of MSVC’s STL library, it is recommended that you use static linkage only. However, it is possible to build libprotobuf and libprotoc as DLLs if you really want. To do this, do the following:

1. Add an additional flag “-DBUILD_SHARED_LIBS=ON” when invoking cmake:

   $ cmake -G “Visual Studio 9 2008” -DBUILD_SHARED_LIBS=ON ..

2. Follow the same steps as described in the above section.
3. When compiling your project, make sure to #define PROTOBUF_USE_DLLS.

When distributing your software to end users, we strongly recommend that you do NOT install libprotobuf.dll or libprotoc.dll to any shared location. Instead, keep these libraries next to your binaries, in your application's own install directory. C++ makes it very difficult to maintain binary compatibility between releases, so it is likely that future versions of these libraries will *not* be usable as drop-in replacements.

If your project is itself a DLL intended for use by third-party software, we recommend that you do NOT expose protocol buffer objects in your library's public interface, and that you statically link protocol buffers into your library.

ZLib support

If you want to include GzipInputStream and GzipOutputStream (google/protobuf/io/gzip_stream.h) in libprotobuf, you will need to do a few additional steps:

1. Obtain a copy of the zlib library. The pre-compiled DLL at zlib.net works.
2. Make sure zlib's two headers are in your include path and that the .lib file is in your library path. You could place all three files directly into this cmake directory to compile libprotobuf, but they need to be visible to your own project as well, so you should probably just put them into the VC shared icnlude and library directories.
3. Add flag “-DZLIB=ON” when invoking cmake:

   ```
   $ cmake -G "Visual Studio 9 2008" -DZLIB=ON ..
   ```

   If it reports NOTFOUND for zlib_include or zlib_lib, you might haven't put the headers or the .lib file in the right directory.
4. Open the generated protobuf.sln file and build as usual.

Notes on Compiler Warnings

The following warnings have been disabled while building the protobuf libraries and compiler. You may have to disable some of them in your own project as well, or live with them.

- C4018 - ‘expression’ : signed/unsigned mismatch
- C4146 - unary minus operator applied to unsigned type, result still unsigned
- C4244 - Conversion from ‘type1’ to ‘type2’, possible loss of data.
- C4251 - ‘identifier’ : class ‘type’ needs to have dll-interface to be used by clients of class ‘type2’
- C4267 - Conversion from ‘size_t’ to ‘type’, possible loss of data.
- C4305 - ‘identifier’ : truncation from ‘type1’ to ‘type2’
- C4355 - ‘this’ : used in base member initializer list
- C4800 - ‘type’ : forcing value to bool ‘true’ or ‘false’ (performance warning)
- C4996 - ‘function’: was declared deprecated

C4251 is of particular note, if you are compiling the Protocol Buffer library as a DLL (see previous section). The protocol buffer library uses templates in its public interfaces. MSVC does not provide any reasonable way to export template classes from a DLL. However, in practice, it appears that exporting templates is not necessary anyway. Since the complete definition of any template is available in the header files, anyone importing the DLL will just end up compiling instances of the templates into their own binary. The Protocol Buffer implementation does not rely on static template members being unique, so there should be no problem with this, but MSVC prints warning nevertheless. So, we disable it. Unfortunately, this warning will also be produced when compiling code which merely uses protocol buffers, meaning you may have to disable it in your code too.