Сборка GDAL 1.9.0 с использованием Visual Studio 2010

Обсудить в форуме Комментариев — 44

Сборка GDAL 1.9.0 с зависимостями

Содержание

Введение

В данной статье пойдет речь о сборке библиотеки GDAL 1.9.0 на Visual Studio 2010.

GDAL - это библиотека для работы с растровыми форматами географических данных. GDAL распространяется Open Source Geospatial Foundation на условиях лицензии X/MIT , то есть является проектом с открытым исходным кодом. Как библиотека GDAL предоставляет вызывающему приложению единую обобщённую модель данных для всех поддерживаемых форматов файлов данных. Помимо этого в состав GDAL входит набор вспомогательных программ, вызываемых из командной строки, для преобразования и обработки данных.[1]

Для сборки GDAL и библиотек от которых она зависит будет использоваться система сборки CMake. CMake (от англ. cross platform make) — это кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода. CMake не занимается непосредственно сборкой, а лишь генерирует файлы управления сборкой из файлов CMakeLists.txt[2]:

- Makefile в системах Unix для сборки с помощью make;

- файлы projects/workspaces в Windows для сборки с помощью Visual C++;
- проекты XCode в Mac OS X

В настоящий момент не существует официального скрипта для сборки GDAL при помощи Cmake. Поэтому, данный метод следует рассматривать в качестве примера сборки и одного из вариантов (унифицированный способ сборки посредством CMake). Кроме того, сборка представляет собой минимальный набор поддерживаемых растровых и векторных драйверов.

Структура библиотеки GDAL

Библиотека GDAL состоит из 3-х основных частей:

1. CPL - базовые функции работы с файловой системой, строками, xml и т.п.
2. OGR - работа с векторными форматами. Список форматов размещен здесь: http://www.gdal.org/ogr/ogr_formats.html
3. GDAL - работа с растровыми форматами. Список форматов размещен здесь: http://www.gdal.org/formats_list.html

OGR и GDAL обеспечивают не только поддержку форматов, но и операции работы с векторами и растрами. Т.е. GDAL позволяет не только считывать но и манипулировать с геоданными.

Про зависимости. Компонент CPL (исходники лежат в подпапке port) зависит от библиотек: zlib, curl.
OGR зависит от многих библиотек, но базовый набор от expat, geos и proj.
GDAL зависит от tiff, jpeg, png.

Следует понимать что такое драйвер. Драйвер - это отдельный плагин обеспечивающий поддержку формата из списков, приведенных выше. В этих списках можно посмотреть, что от чего зависит. Иногда драйвер не зависит ни от какой из сторонних библиотек и компилировать его просто. Иногда нужна еще отдельная библиотека или целый набор библиотек.

О сборке. В зависимости от необходимости можно собрать GDAL именно в той конфигурации, что нужна (с определенным перечнем драйверов). Собирать будем пошагово: вначале все зависимые библиотеки, затем сам GDAL. Для простоты предлагается делать это все с использованием системы сборки CMake. Для большинства библиотек уже имеются скрипты CMake и сборка становится однотипной и простой. Заодно на выходе будут привычные проекты Visual Studio, где можно зайти в свойства и посмотреть, что и как.

Поэтому скачиваем последнюю версию системы сборки CMake по адресу http://www.cmake.org/cmake/resources/software.html (на текущий момент последняя версия http://www.cmake.org/files/v2.8/cmake-2.8.7-win32-x86.exe) и ставим ее.

Примечание. CMake, как, впрочем, и другие системы сборки, поддерживает два способа сборки: при первом (**In source**, in tree build) целевая программа собирается в тех же директориях, в которых расположены исходные тексты. В результате в директориях с исходниками остаются промежуточные файлы сборки. Второй вариант - сборка **out of source**, при которой сборка выполняется в отдельной директории, в которой сохраняются все промежуточные файлы, а директория с иходными текстами остается в том же состоянии, что и до выполнения сборки.[3]

В предложенной методике будет применяться сборка Out-of-source.

Сборка зависимых библиотек

Библиотека expat

Первой библиотекой будем компилировать **expat**.

Скачиваем исходные тексты библиотеки отсюда http://sourceforge.net/projects/expat/files/expat/2.0.1/ (актуальная версия на момент написания находится в файле под названием expat-2.0.1.tar.gz) и распаковываем в папку, где будут храниться все проекты для сборки библиотек. Например, по следующему пути

```
C:/project/expat-2.0.1/
```

Обратите внимание, что эта и все последующие папки должны быть разархивированы так, что в конечной папке, в данном случае expat-2.0.1, должны лежать исходные тексты и сопровождающие файлы (см. рис. 1), а не вложенная папка expat-2.0.1 (для данного примера).

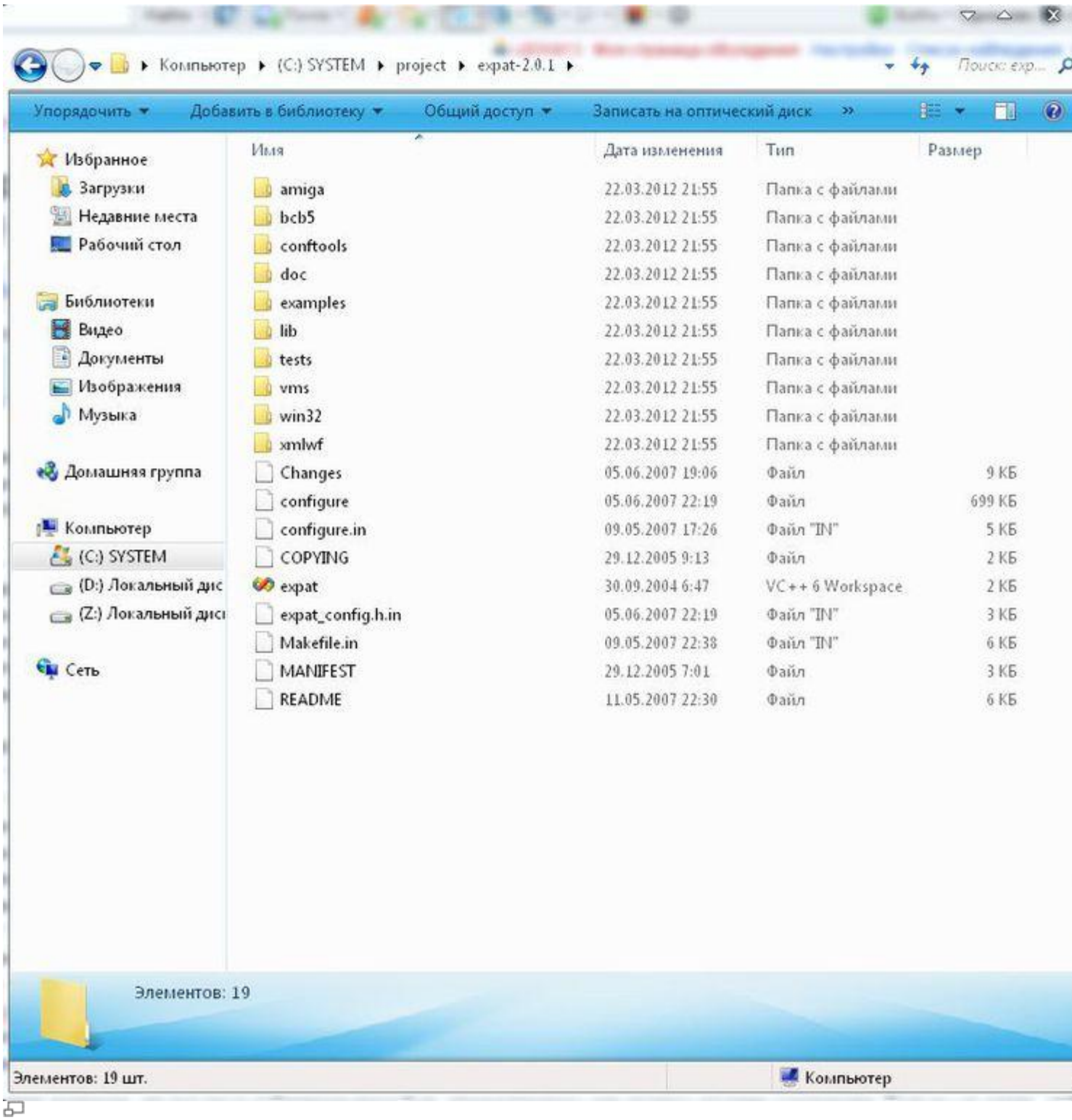

Рис. 1. Структура директории

Далее скачиваем файл CMakeLists.txt (Файл:CMakeLists.zip) и помещаем в разархивированную папку expat-2.0.1.

Запускаем CMake GUI и указываем два пути: *Where is the source code* - это путь куда был распакован expat, например *C:/project/expat-2.0.1* и *Where to build the binaries* – это папка, куда генерируются файлы проекта *C:/project/expat-2.0.1/build*. Нажимаем кнопку *Configure* и выбираем версию Visual Studio (в нашем случае Visual Studio 2010), если выдаст ошибку, нажимаем еще раз. После нажимаем кнопку Generate.

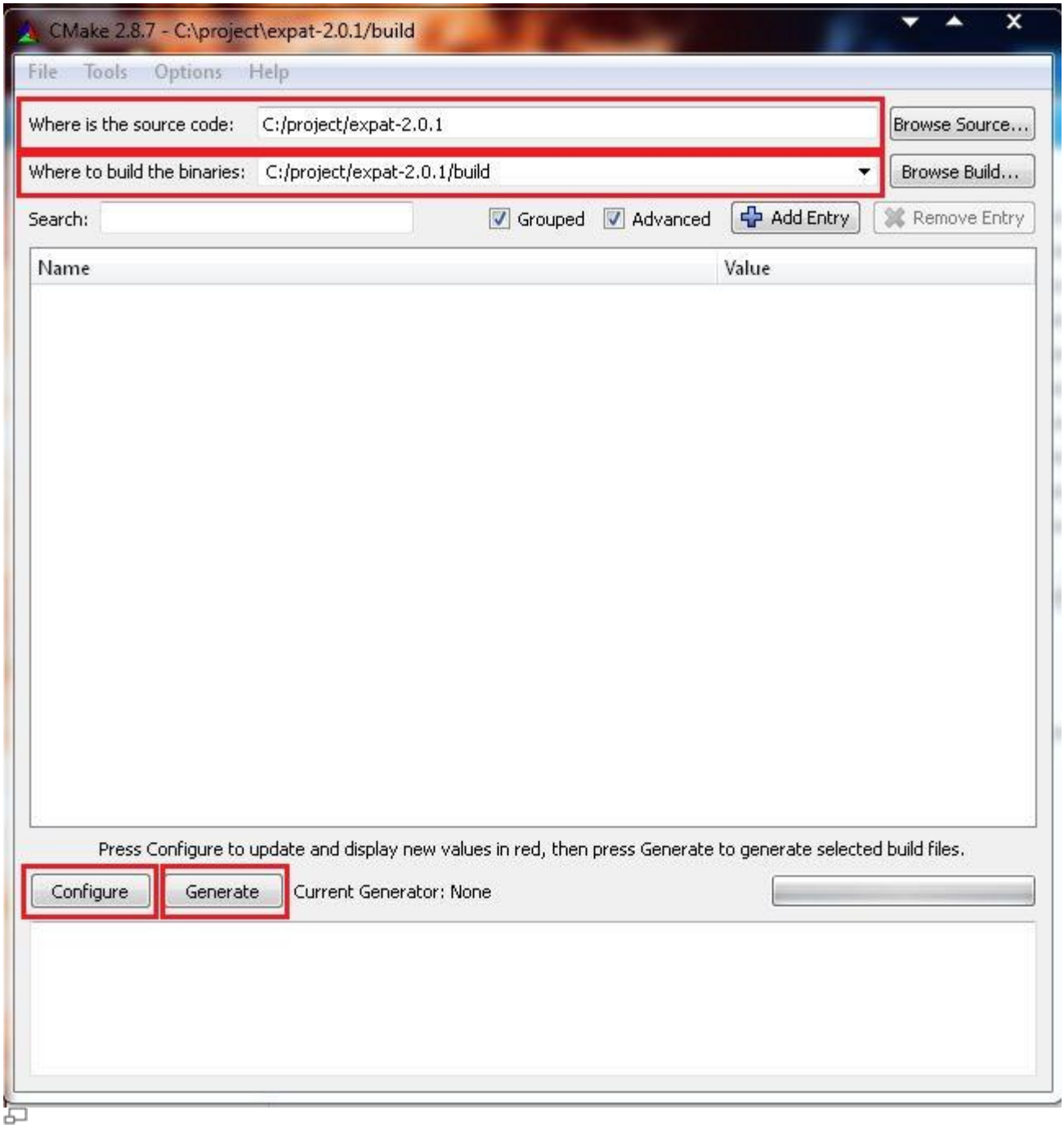

Рис. 2. Окно интерфейса пользователя Cmake

В папке *build* должен появиться фаил *libexpat.sln*. Открываем его с помощью Visual Studio, выставляем сверху Тип сборки *Release* вместо *Debug* и компилируем проект (нажимаем F7).

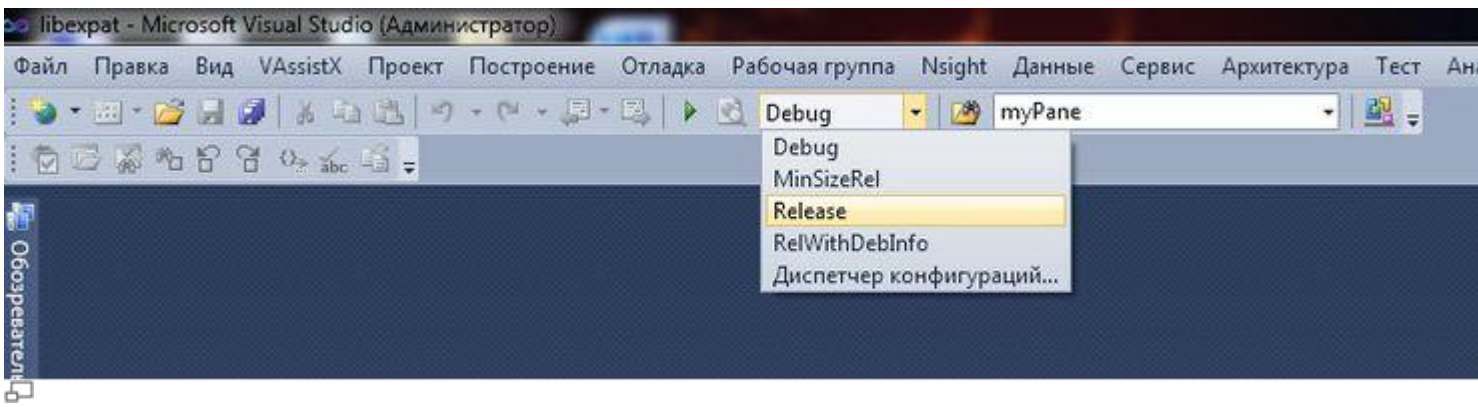

Рис. 3. Выбор типа сборки

После этого в папке *C:/projects/expat-2.0.1/lib/Release* (далее все скомпилированные библиотеки будем класть в схожие папки *lib/Release*) должны появиться следующие файлы

- libexpat.dll
- libexpat.exp
- libexpat.lib

Это был скомпилирован так называемая **shared dll**. Существует еще **static dll** - тогда на выходе получается только *libexpat.lib*.

Потом *libexpat.dll* надо будет скопировать в папку с вашей программой. Туда же надо будет скопирвоать все зависимые библиотеки.

Библиотека zlib

Следующей библиотекой будет **zlib**.

Скачиваем ее отсюда http://zlib.net/ (актуальная на момент написания статьи версия http://zlib.net/zlib126.zip). Файл сценария CMake - CmakeLists.txt в архив уже включен. Распакуем по следующему пути: *C:/project/zlib-1.2.6*.

При нажатии на кнопку Configure в самом конце cmake напишет следующее сообщение:

```
CMake Error at CMakeLists.txt:65 (message):
 You must remove C:/project/zlib-1.2.6/zconf.h from the
 source tree.  This file is included with zlib but CMake generates this file
 for you automatically in the build directory.
```

Здесь говорится, что нужно стереть файл *C://project/zlib-1.2.6/zconf.h* т.к. он сценарий CMake генерирует новый файл. После удаления zconf.h опять нажимаем *Configure*. Процесс должен закончиться без ошибок. Далее запускаем *Generate* и переходим в папку *build*. Открываем файл *zlib.sln* с помощью Visual Studio (далее VS), ставим тип сборки *Release*. А так же заходим в меню в пункт Построение и выбираем диспетчер конфигураций, после этого **снимаем галочки** с *examles* и *minizip* они нам не понадобятся (см. рис. 4), если же будут нужны, необходимо в свойствах этих проектов прописать путь к *zlib.h*.

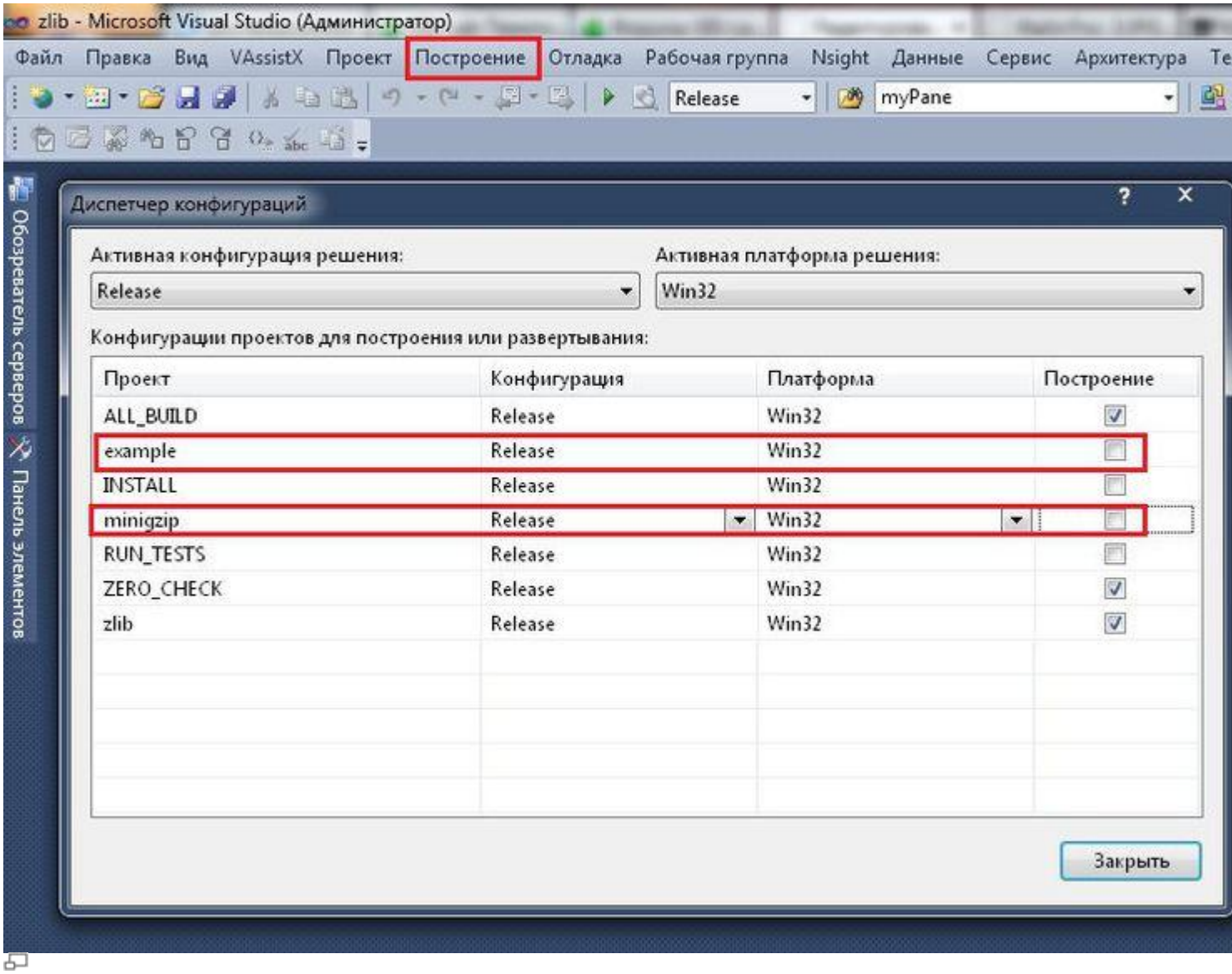

Рис. 4. Диспетчер конфигураций

После всего этого компилируем проект и получаем 3 файла в папке *C:/project/zlib-1.2.6/Debug/Release*. Перенесем их в папку *C:/project/zlib-1.2.6/lib/Release* для унификации (эту папку необходимо создать самим).

Библиотека geos

Следующей библиотекой будет **geos**.

Скачиваем отсюда http://trac.osgeo.org/geos/ (актуальная версия на момент написания статьи находится по адресу http://download.osgeo.org/geos/geos-3.3.2.tar.bz2). Извлекаем в *C:/project/geos-3.3.2* . Делаем конфигурацию в CMake, после этого в разделе Ungrouped Entries убираем галочку с BUILD_TESTING (см. рис. 5).

Примечание, чтобы у вас появился раздел Ungrouped Entries, поставьте галки в Grouped и Advanced (см. рис. 5).

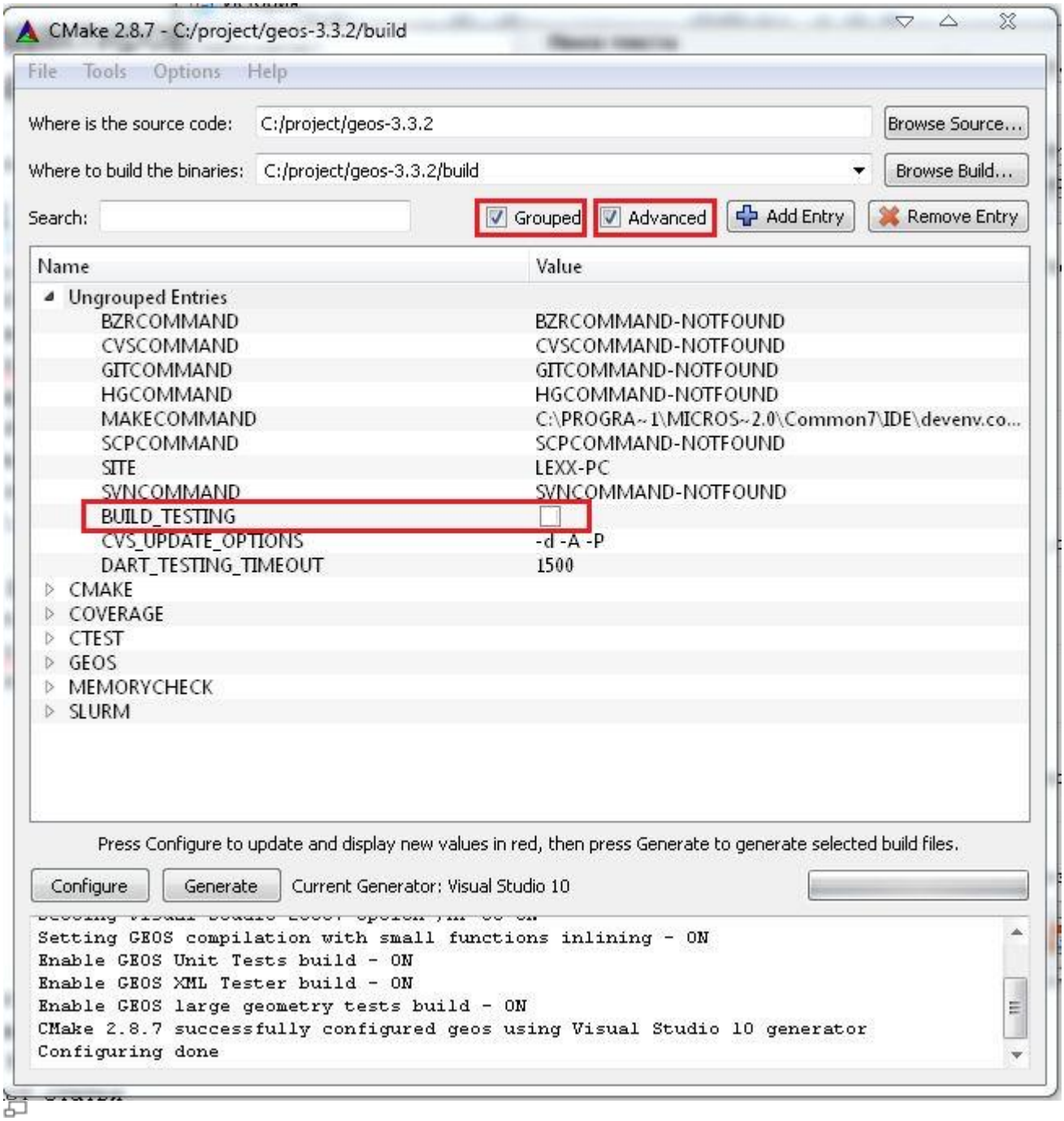

Рис. 5. Настройки GEOS в CMake-GUI

Далее нажимаем Generate, открываем в VS файл *C:/project/geos-3.3.2/build/geos.sln* , выставляем конфигурацию *Release* и компилируем. Если у вас возникла ошибка:

```
Ошибка 1 error C2668: log: неоднозначный вызов перегруженной функции C:/project/geos-3.3.2/src/operation/buffer/BufferOp.cpp       97 1 geos-static
```

кликните по ней 2 раза мышкой и в открывшемся файле замените строчку

```
int bufEnvPrecisionDigits = (int) (std::log(bufEnvMax) / std::log(10) + 1.0);
```

на

```
int bufEnvPrecisionDigits = (int) (std::log(bufEnvMax) / std::log(10.0) + 1.0);
```

и скомпилируйте заново.

Далее, если возникла ошибка

```
Ошибка 1 error C2039: toupper: не является членом "std"
        C:/project/geos-3.3.2/tests/unit/capi/GEOSisValidDetailTest.cpp      60 1
geos_unit
```

кликните по ней 2 раза мышкой и в открывшемся файле сверху припишите #include <ctype.h>, а вместо

```
str[i] = std::toupper(str[i]);
```

напишите

```
str[i] = toupper(str[i]);
```

и скомпилируйте еще раз. После того как все закончится успешно скопируйте файлы из папки *C:/project/geos-3.3.2/build/lib/Release* в папку *C:/project/geos-3.3.2/lib/Release*.

Библиотека curl

Следующей библиотекой будет **curl**. Скачиваем последнюю версию отсюда http://curl.haxx.se/download.html (актуальная версия на момент написания статьи находилась по следующему пути http://curl.haxx.se/download/curl-7.24.0.zip) и распакуем в *C:/project/curl-7.24.0*.

Запускаем CMake GUI, указываем пути и нажимаем Configure. Далее прописываем в свойствах

- WSOCK32_LIBRARY - WSOCK32.lib
- WS2_32_LIBRARY - WS2_32.lib

в ZLIB_INCLUDE_DIR прописываем два пути через точку запятую:
C:/project/zlib-1.2.6;C:/project/zlib-1.2.6/build
в ZLIB_LIBRARY прописываем путь до zlib.lib
C:/project/zlib-1.2.6/lib/Release/zlib.lib
И последнее расставляем галочки как на скриншоте (см. рис. 6).

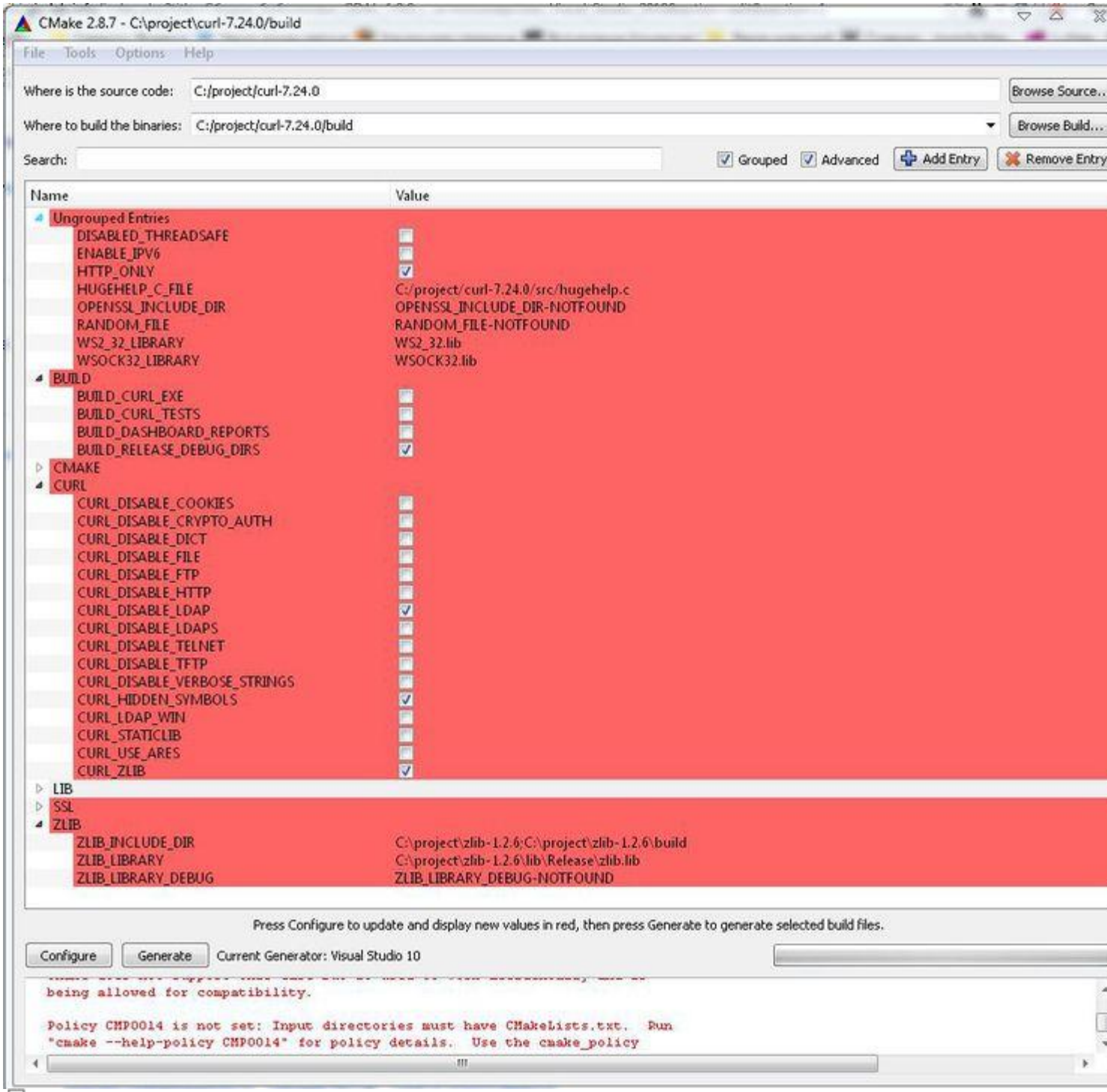

Рис. 6. Настройки CURL в CMake-GUI

После того как все будет сделано, нажимайте *Configure*, все должно закончиться без ошибок и потом *Generate*. Далее открываем файл *CURL.sln* в папке *build*. Заходим в свойства проекта (для этого в Обозревателе решений надо выбрать libcurl)(Проект -> Свойства), далее Свойства конфигурации -> Компоновщик -> Ввод, далее в окне справа Дополнительные зависимости, нажимаем Изменить и удаляем:

- WSOCK32_LIBRARY-NOTFOUND
- WS2_32_LIBRARY-NOTFOUND

Нажимаем применить и ОК, далее компилируем проект. После успешной компиляции помещаем файлы из папки *C:/project/curl-7.24.0/build/lib/Debug* в папку *C:/project/curl-7.24.0/lib/Realese*

Библиотека libjpeg

Далее будем собирать libjpeg - брать здесь http://www.ijg.org/ последнюю версию http://www.ijg.org/files/jpegsr8d.zip . CMakeLists скачиваем тут Файл:CMakeListsjpeg.zip и разархивируем,

кладем в папку *C:/project/jpeg-8d*.
Находим файл jmorecfg.h и меняем строку 197 строчку на:

```
//#define EXTERN(type)          extern type
#ifdef _WIN32
#    ifdef libjpeg_EXPORTS
#        define EXTERN(type) extern /*"C"*/ __declspec(dllexport) type
#    else
#        define EXTERN(type) extern /*"C"*/ __declspec(dllimport) type
#    endif
#else
#    define EXTERN(type)               extern type
#endif
```

Запускаем CMake и делаем *configurate*, затем *Generate*. Идем в папку *Build*, открываем *libjpeg.sln*, ставим *Release* и компилируем.

Библиотека libpng

Следующей библиотекой будет **libpng**. Скачиваем последнюю версию отсюда http://www.libpng.org/pub/png/libpng.html (актуальная версия на момент написания статьи находилась по следующему пути http://download.sourceforge.net/libpng/lpng159.zip). Распаковываем и запускаем CMake GUI, проводим конфигурацию. Прописываем пути для ZLIB_INCLUDE_DIR и ZLIB_LIBRARY-NOTFOUND как делали это выше. Нажимаем Configure и Generate. Далее открываем libpng.sln, ставим конфигурацию Release и компилируем. Файлы из папки *C:/project/lpng159/build/Release* переносим в *C:/project/lpng159/lib/Release* .

Библиотека libtiff

Следующей библиотекой будет **libtiff**. Скачиваем последнюю версию отсюда ftp://ftp.remotesensing.org/pub/libtiff (актуальная версия на момент написания статьи находилась по следующему пути ftp://ftp.remotesensing.org/pub/libtiff/tiff -4.0.1.zip), распаковываем в *C:/project/tiff-4.0.1*, качаем CMakeLists и еще пару файлов и распаковываем как обычно Файл:Tiff.zip. Запускаем CMake, делаем *Configure* и прописываем пути для ZLIB_INCLUDE_DIR и ZLIB_LIBRARY-NOTFOUND как делали это выше. Нажимаем *Configure* еще раз и *Generate*, далее открываем *libtiff.sln* ставим конфигурацию *Release* и компилируем.

Библиотека proj

Следующей библиотекой будет **proj**. Скачиваем последнюю версию отсюда http://trac.osgeo.org/proj/ (актуальная версия на момент написания статьи находилась по следующему пути http://download.osgeo.org/proj/proj-4.8.0.zip). Распаковываем, и кидаем туда этот файл Файл:CMakeListsproj.zip, запускаем CMake, прописываем пути до proj, жмем *Configure* и *Generate*, открываем *proj4.sln*, ставим конфигурацию *Release* и компилируем.

На этом вся подготовка закончена, у нас есть все необходимые библиотеки.

Сборка GDAL

Приступаем к сборке самого **GDAL**.

Скачиваем отсюда http://trac.osgeo.org/gdal/wiki/DownloadSource последнюю версию на момент написания статьи gdal 1.9.0 (http://download.osgeo.org/gdal/gdal190.zip). Качаем архив Файл:Gdal trank.zip . Распаковываем gdal в папку *project*. В полученную папку *C:/project/gdal-1.9.0* распаковываем файлы из архива *Gdal_trank.zip* (папку cmake и файл CMakeLists).

Либо скачиваем последнюю версию CMake файлов для GDAL вместе с самим GDAL с GitHub:

```
git clone https://github.com/aashish24/gdal-svn.git -b cmake4gdal gdal-cmake-trunk
```

И получаем папку gdal-cmake-trunk.

Запускаем CMake и как обычно прописываем 2 пути *C:/project/gdal-1.9.0 и C:/project/gdal-1.9.0/build*, нажимаем *Configure*.
Когда выскочит ошибка, прописываем пути до **zlib**: *C:/project/zlib-1.2.6;C:/project/zlib-1.2.6/build* и *C:/project/zlib-1.2.6/lib/Release/zlib.lib* в директориях соответственно ZLIB_INCLUDE_DIR и ZLIB_LIBRARY. Далее все пути надо будет прописывать примерно аналогично для директории INCLUDE и LIBRARY.
Нажимаем *Configure*, получаем ошибку и прописываем пути для **CURL** *C:/project/curl-7.24.0/include* и *C:/project/curl-7.24.0/lib/Release/libcurl_imp.lib* .
Нажимаем *Configure*, получаем ошибку и прописываем пути для **EXPAT**: *C:/project/expat-2.0.1/lib/* и *C:/project/expat-2.0.1/lib/Release/libexpat.lib* .
Нажимаем *Configure*, получаем ошибку и прописываем пути для **GEOS**: *C:/project/geos-3.3.2/include* и *C:/project/geos-3.3.2/lib/Release/geos_c.lib* .
Нажимаем *Configure*, получаем ошибку и прописываем пути для **PROJ**: *C:/project/proj-4.8.0/src* и *C:/project/proj-4.8.0/lib/Release/proj4.lib* .
Нажимаем *Configure*, получаем ошибку, идем в **OGR** и убираем галочку с OGR_PG.
Нажимаем *Configure*, получаем ошибку и прописываем пути для **JPEG**: *C:/project/jpeg-8d;C:/project/jpeg-8d/build* и *C:/project/jpeg-8d/lib/Release/libjpeg.lib* .
Нажимаем *Configure*, получаем ошибку и прописываем пути для **TIFF**: *C:/project/tiff-4.0.1/libtiff;C:/project/tiff-4.0.1/build* и *C:/project/tiff-4.0.1/lib/Release/libtiff.lib* .
Нажимаем *Configure*, получаем ошибку и прописываем пути для **PNG**, **тут внимательно**! Первая строчка - пути до LIBRARY: *C:/project/lpng159/lib/Release/libpng15.lib* , вторая *C:/project/lpng159;C:/project/lpng159/build* .
Нажимаем *Configure*, все должно закончиться успешно, нажимаем *Generate*. Далее идем в папку *C:/project/gdal-1.9.0/build* и открываем файл *gdallib.sln* с помощью VS. Ставим конфигурацию *Release* и компилируем. Если все прошло без ошибок, **вы собрали GDAL!**

Теперь необходимо проверить работоспособность собранной библиотеки.

Тестовый пример

Сделаем тестовый пример. Исходный код берем отсюда: http://www.gdal.org/gdal_tutorial_ru.html .
Создадим консольный проект C++ CLR. Пропишем следующие пути:
выбираем в меню Проект->Свойства...->Свойства конфигурации->Компоновщик->Ввод. На этой странице в дополнительных зависимостях прописываем *C:/project/gdal-1.9.0/build/Release/gdal19.lib*
теперь в Проект->Свойства...->Свойства конфигурации->C/C++ в Дополнительные зависимости включаемых файлов прописать

- *C:/project/gdal-1.9.0/build/port/*
- *C:/project/gdal-1.9.0/alg/*
- *C:/project/gdal-1.9.0/*
- *C:/project/gdal-1.9.0/*gcore
- *C:/project/gdal-1.9.0/*port

Так же необходимо в папку, где содержатся откомпилированные файлы проекта (обычно папка с именем конфигурации *Release* или *Debug*) положить сформированные dll, собрынные нами ранее:

- gdal19.dll
- geos.dll
- geos_c.dll
- libcurl.dll
- libexpat.dll
- libjpeg.dll
- libpng15.dll
- libtiff.dll
- proj4.dll
- zlib1.dll

Так же в эту папку положите какой-нибудь рисунок и задайте путь до него в тексте программы. Например,

1.JPG.

Так выглядит тестовый код:

```
#include "stdafx.h"
#include "gdal_priv.h"
using namespace System;
int main(array<System::String ^> ^args)
{
    GDALDataset  *poDataset;
    GDALAllRegister();
    poDataset = (GDALDataset *) GDALOpen( "1.JPG", GA_ReadOnly ); // здесь надо указать
путь до изображения
         if( poDataset == NULL )
    {
       return 0;
    }
        double        adfGeoTransform[6];
    printf( "Драйвер: %s/%s\n",
            poDataset->GetDriver()->GetDescription(),
            poDataset->GetDriver()->GetMetadataItem( GDAL_DMD_LONGNAME ) );
    printf( "Размер %dx%dx%d\n",
            poDataset->GetRasterXSize(), poDataset->GetRasterYSize(),
            poDataset->GetRasterCount() );
    if( poDataset->GetProjectionRef()  != NULL )
        printf( "Проекция \"%s\"\n", poDataset->GetProjectionRef() );
    if( poDataset->GetGeoTransform( adfGeoTransform ) == CE_None )
    {
        printf( "Начало координат (%.6f,%.6f)\n",
                adfGeoTransform[0], adfGeoTransform[3] );
        printf( "Размер пиксела (%.6f,%.6f)\n",
                adfGeoTransform[1], adfGeoTransform[5] );
    }
    return 0;
}
```

Заключение

В статье мы научились собирать библиотеку **GDAL** и библиотеки от которых она зависит, а также сделали тестовый пример для работы с ней. Комментарии, исправления, отзывы и пожелания **приветствуются**.

1. ↑ GDAL - Geospatial Data Abstraction Library
2. ↑ CMake, Материал из Википедии — свободной энциклопедии
3. ↑ CMake, Сборка In-source и Out-of-source

Огромное СПАСИБО за помощь со сборкой и терпение пользователю ГИС-ЛАБ Bishop.

Обсудить в форуме Комментариев — 44

Последнее обновление: 2014-05-15 00:08

Дата создания: 20.03.2012

Автор(ы): Евгений Боровенский (LEXX413), Дмитрий Барышников