GDAL can be built on Win32 platforms using MS VC++ 6.x and MS Visual Studio .NET (C++). The source distribution includes NMAKE compatible makefiles for use from the command shell. For those wanting Visual Studio 7.1 or 8.0 Solution files (instead of the command line nmake files), Ben Discoe normally makes them available on the VTP site. Only the nmake file approach is directly supported by the GDAL maintainers.

Regarding use of Visual Studio projects check details on:

- Generating Visual Studio Project for VS 2010 or later.
- Generating Visual Studio 2003 Project

Windows CE is also supported by an alternate set of build files. See GDAL for Windows CE for details.

Prebuild Libraries

A prebuilt GDAL 1.4.1 library is available at:

http://download.osgeo.org/gdal/win32/1.4.1/gdalwin32dev141.zip

This should be unpacked on top of the executable package which includes the gdal14.dll:

http://download.osgeo.org/gdal/win32/1.4.1/gdalwin32exe141.zip

The *dev* package includes:

- Include files in the gdalwin32-1.4.1\include directory.
- An release (/MD /Ox) DLL (bin\gdal14.dll) and stub library (lib\gdal_i.lib).

The DLL and stub libraries are built with Visual Studio.NET 2003 (VC 7.1), which is likely required for any use of the C++ API. The libraries should be compatible with any win32 C/C++ compiler if only the GDAL/OGR C API is used. No debug version is offered.

Build SDKs

A complete set of the files required to build GDAL with MSVC2003/2005/2008 for Win32/Win64 are available to download at:

http://download.gisinternals.com/sdk.html

Please refer to the readme.txt in the packages for further build instructions.

Building From Source

First you should check the basic options in nmake.opt especially the settings for the VC variant to use and for the installation directory.

For command line builds you will normally have to have run the VCVAR32.BAT script that comes with the compiler. For VC 6.x this might be found at:

```
C:\Program Files\Microsoft Visual Studio\VC98\bin\VCVARS32.BAT
```

Hint: to build for x64, you may want to add this line in a bat file (shown for VS2010):

```
call "C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC\vcvarsall.bat" am
```

Some Visual Studios also provide a "Visual Studio Command Prompt" to be found in the start menu, which gives you a shell with the correct environment. If you plan to compile for the 64bit platform be sure to choose the correct bat / command prompt.

Once the environment is setup, you can cd to the GDAL root directory and do the following:

```
C:\GDAL> nmake /f makefile.vc
```

Once the build has completed successfully, you can also install the required GDAL files for using GDAL utilities using the install makefile target. Ensure that BINDIR, and DATADIR are set appropriately in the nmake.opt file before doing this.

```
C:\GDAL> nmake /f makefile.vc install
```

If you wish to build your own applications using GDAL you can use the following command to install all the required libraries, and include files as well. Ensure that LIBDIR and INCDIR are properly set in the nmake.opt file.

```
C:\\GDAL> nmake /f makefile.vc devinstall
```

Projects linking against GDAL should include the directory specified in INCDIR in the include path, and the directory specified in LIBDIR in their /LIBPATH. To use gdal link against the gdal_i.lib stub library.

Basic Options

There are lots of options that can be tweaked by hand in the nmake.opt file in the GDAL root directory.

- **MSVC_VER**: Version of Visual C++ compiler to use. Details on version naming are mentioned in nmake.opt.
- **WIN64**: Enable this if compiling for the 64 bit platform. You also need to use the correct Visual C++ compiler and set the environment properly.
- **GDAL_HOME**: Directory in which to install GDAL. This is the default prefix for all following directory settings.
- **BINDIR**: Directory in which to install executables, and DLLs if you use the "nmake /f makefile.vc install" command.
- **LIBDIR**: Directory in which to install stub library (gdal_i.lib).
- **INCDIR**: Directory in which to install GDAL/OGR include files.
- **DATADIR**: Directory in which to install GDAL required data files.
- **OPTFLAGS**: Set this to contain the desired compiler flags for compiling. The default is a debug build but there is usually a suitable alternative for optimized builds commented out in the template file. Please, note that if you will build GDAL against the third-party libraries (installed from precompiled binaries or compiled by yourself) you should keep the run-time linking options for that libraries and for GDAL the same. For example, if you will use /MD option in **OPTFLAGS** string (link against multithreaded dynamic library), the same should be used to build other libraries, such as HDF4 or JasPer. If you don't have library sources then read the documentation, supplied with library, to figure out how it was compiled and set up the same linking option in **OPTFLAGS**. Mixing different linking options or release/debug flags will result in build failures or in crashes during run time.

Advanced Options

- **PY_INST_DIR**: Directory to install python modules in. Only required if Python support is being built.
- **PYDIR**: The directory where your python tree is installed. Used to find python include files. If this is set to a directory that does not exist then python support will be omitted from the

build.

- **DLLBUILD**: Define this (to "1") to force all the OGR utilities to link against the GDAL DLL instead of building built statically.
- **INCLUDE_OGR_FRMTS**: Set to "YES" to build OGR formats into the GDAL dll, or comment out to omit OGR formats from build.
- **SETARGV**: Point this to the setargv.obj file distributed with Visual Studio to get wildcard expansion for command line arguments or leave commented out to omit this.
- **ECWDIR/ECWLIB**: Uncomment these to enable building with ECW support, correcting ECWDIR to point to the install location on your system.
- **OGDIDIR/OGDIVER/OGDILIB**: Uncomment these to enable OGDI support, correcting the OGDIDIR and OGDIVER values as needed.
- **HDF4_DIR**: Uncomment, and correct path to enable support for NCSA HDF Release 4.
- **JASPER_DIR/JASPER_INCLUDE/JASPER_LIB**: This variables should be pointed to appropriate directories where JasPer library was installed. JasPer toolkit needed for JPEG2000 support.
- **XERCES_DIR/XERCES_INCLUDE/XERCES_LIB**: Uncomment these and correct XERCES_DIR to enable Xerces XML parser support for GML read support.
- **FME_DIR**: Uncomment, and correct path to enable support for FMEObject vector access.
- **JPEG_EXTERNAL_LIB/JPEGDIR/JPEG_LIB**: Used to be able to link gdal with an external JPEG library. Uncomment these lines and correct the paths.
- **PNG_EXTERNAL_LIB/PNGDIR/PNG_LIB**: Used to be able to link gdal with an external PNG library. Uncomment these lines and correct the paths.
- **DODS_DIR/DODS_LIB**: Uncomment these lines and correct the paths to enable DODS/OPeNDAP support. Currently this is building with libdap 3.8.x. You'll also need the GNU regex.h for win32.

I frequently forget to update the Windows makefiles when I add new files, so if something comes up missing consider comparing the file lists in the appropriate makefile.vc against the GNUmakefile or just contact me.

Building Plugins

Plugins are one way you can add support for format drivers without embedding object code that links to auxiliary libraries that can cause linking issues (usually licensing related). Currently, four drivers support building as plugins:

- MrSID
- Oracle Spatial (raster and vector)
- ArcSDE (raster and vector)
- SOSI (vector)

Plugin building example

This example allows the building of ArcSDE support as a plugin. To build SDE support as a plugin, build GDAL as you normally would without enabling any of the SDE-specific options in nmake.opt. After you have a gdal.dll, go back and edit nmake.opt to the following:

```
        SDE_ENABLED = YES # turn on support
        SDE_VERSION = 91
        SDE_PLUGIN = YES
        SDE_SDK = C:\arcgis\arcsde # base directory of SDE includes and librar
        SDE_INC = $(SDE_SDK)\include  # includes directory
        SDE_LIB = $(SDE_SDK)\lib\pe$(SDE_VERSION).lib \
                  $(SDE_SDK)\lib\sde$(SDE_VERSION).lib $(SDE_SDK)\lib\sg$(SDE_
```

Note: if the SDE_PLUGIN variable is not set, but all of the other SDE-related variables are, the SDE will be linked in directly like any other auxiliary library. After editing nmake.opt, cd into frmts/sde using your MSVC terminal and issue the make command:

```
        nmake /f makefile.vc plugin
```

This will build gdal_SDE.dll and link it against the gdal.dll and the SDE SDK client DLLs. After you have your gdal_SDE.dll, copy it into your GDAL_DRIVER_PATH folder (you will need to explicitly set this environment variable to allow GDAL to find its plugin drivers). Your next gdalinfo or gdal_translate (or any other GDAL calls) should now be able to see the driver when it goes through the driver registration process.

Last modified on Feb 16, 2017, 2:04:03 PM