Installing From Source Code

Applicability

This section provides instructions for building GraphicsMagick Microsoft Windows using the Visual C++ (Visual Studio, etc.) IDE. For building using the free Cygwin or MinGW compilers, follow the instructions in INSTALL-unix.txt.

Important Notes

On some platforms Visual C++ may fail with an internal compiler error. If this happens to you, then make sure that your compiler is updated with the latest fixes from the Microsoft web site and the problem should go away.

Windows WIN2K/98 Visual C++ 6.0, 7.X, and 8.0 Compilation

The Visual C++ distribution targeted at Windows XP, Win2K, or Windows 98 does not provide any stock workspace (DSW) or project files (DSP) except for those included with third party libraries. Instead, there is a "configure" program that must run to create build environments to satisfy various requirements.

The Visual C++ system provides three different types of "runtimes" that must match across all application, library, and DLL code that is built. The "configure" program creates a set of build files that are consistent for a specific runtime selection.

The three options for runtime support are:

1. Dynamic Multi-threaded DLL runtimes (VisualDynamicMT).
2. Static Single-threaded runtimes (VisualStaticST).
3. Static Multi-threaded runtimes (VisualStaticMT).
4. Static Multi-threaded DLL runtimes (VisualStaticMTDLL).

In addition to these runtimes, the VisualMagick build environment allows you to select whether to include the X11 libraries in the build or not. X11 DLLs and headers are provided with the VisualMagick build environment. Most Windows users do not use X11 so they will prefer to build without X11 support. When X11 is not supported, gm subcommands 'animate', 'display', and 'import' will not work.

This leads to five different possible build options, which should cover almost any particular situation. The default binary distribution is built using #1 from above with the X11 libraries included. This results in an X11 compatible build using all DLL's for everything and multi-threaded support (the only option for DLL's).

To do a build for your requirements, simply go to the configure sub- directory under VisualMagick and open the configure.dsw workspace (for Visual C++ 6.0) or configure.sln (for Visual C++ 7.X or 8.X). Set the build configuration to "Release" under the

"Build..., Set Active Configuration..." menu.

Build and execute the configure program and follow the on-screen instructions. You should not change any of the defaults unless you have a specific reason to do so.

The configure program provides a button entitled

Edit "magick_config.h"

Clicking this button brings up magick_config.h in Windows notepad for optionally changing any preprocessor defines in GraphicsMagick's magick_config.h file. This file is copied to `magick\magick_config.h`. You may safely open`magick\magick_config.h`, modify it, and recompile without re-running the configure program. In fact, using notepad to edit the copied file may be preferable since it preserves the original magick_config.h file.

Key user tunables in magick_config.h include:

QuantumDepth (default 8)

Specify size of PixelPacket color Quantums (8, 16, or 32) A value of 8 uses half the memory than 16 and may run 30% faster, but provides 256 times less color resolution than a value of 16.

UseInstalledMagick (default undefined)

Define to build a GraphicsMagick which uses registry settings or embedded paths to locate installed components (coder modules and configuration files). The default is to look for all files in the same directory as the executable.

ProvideDllMain (default undefined)

Define to include a DllMain() function ensures that the GraphicsMagick DLL is properly initialized without participation from dependent applications. This avoids the requirement to invoke IntializeMagick() from dependent applications but only works for DLL builds.

After creating your build environment you can proceed to open the DSW (or SLN) file that was generated in the VisualMagick directory and build everything from there.

In the final DSW file you will find a project call "All". In order to build everything in the distribution, select this project and make it the "active" project. Set the build configuration to the desired one (Debug, or Release) and do a "clean" followed by a "build". You should do the build in a specific way:

1. Make the "All" project the active project (Bold) Right click on the All project and select "Set As Active Project"
2. Select "Build..., Clean"
3. Select "Build..., Build"
4. Go get some coffee unless you have a very fast machine!.

The "Clean" step is needed in order to make sure that all of the target support libraries are updated with any patches needed to get them to compile properly under Visual C++.

All of the required files that are needed to run any of the command line tools will be found in the "bin" subdirectory of the VisualMagick subdirectory. This includes EXE, and DLL files. You should be able to test the build directly from this directory without having to move anything to any of the global SYSTEM or SYSTEM32 areas in the operating system installation.

Note #1:

The Visual C++ distribution of GraphicsMagick comes with the Magick++ C++ wrapper by default. This add-on layer has a large number of demo and test

files that can be found in `GraphicsMagick\Magick++\demo`,
and `GraphicsMagick\Magick++\tests`. There are also a variety of tests that use the straight C API as well in `GraphicsMagick\tests`.

All of these programs are NOT configured to be built in the default workspace created by the configure program. You can cause all of these demos and test programs to be built by checking the box in configure that says:

"Include all demo and test programs"

In addition, there is another related checkbox (checked by default) that causes all generated project files to be created standalone so that they can be copied to other areas of you system.

This is the checkbox:

"Generate all utility projects with full paths rather then relative paths"

WOW - that a mouthfull - eh?

Visual C++ uses a concept of "dependencies" that tell it what other components need to be build when a particular project is being build. This mechanism is also used to ensure that components link properly. In my normal development environment, I want to be able to make changes and debug the system as a whole, so I like and NEED to use dependencies. However, most end users don't want to work this way.

Instead they really just want to build the package and then get down to business working on their application. The solution is to make all the utility projects (UTIL_xxxx_yy_exe.dsp) use full absolute paths to all the things they need. This way the projects stand on their own and can actually be copied and used as templates to get a particular custom application compiling with little effort.

With this feature enabled, you should be able to nab a copy of...

`VisualMagick\utilities\UTIL_convert_xxx_exe.dsp` (for C)

-or-

`VisualMagick\Magick++\demo\UTIL_demo_xxx_exe.dsp` (for C++)

... and pop it into notepad, modify it (carefully) to your needs and be on your way to happy compiling and linking.

You can feel free to pick any of the standard utilities, tests, or demo programs as the basis for a new program by copying the project and the source and hacking away.

The choice of what to use as a starting point is very easy...

For straight C API command line applications use something from

`GraphicsMagick\tests` or `GraphicsMagick\utilities` (source code)

`GraphicsMagick\VisualMagick\tests` or `GraphicsMagick\Visualmagick\utilities` (project - DSP)

For C++ and Magick++ command line applications use something from

`GraphicsMagick\Magick++\tests` or `GraphicsMagick\Magick++\demo` (source code)

`GraphicsMagick\VisualMagick\Magick++\tests` or `GraphicsMagick\VisualMagick\Magick++\demo` (project - DSP)

For C++ and Magick++ and MFC windows applications use

`GraphicsMagick\win2k\IMDisplay` (source code)

`GraphicsMagick\VisualMagick\win32\NtMagick` (project - DSP)

Note #2:

The GraphicsMagick distribution is very modular. The default configuration is there to get you rolling, but you need to make some serious choices when you wish to change things around.

The default options are all targeted at having all the components in one place (e.g. the "bin" directory of the VisualMagick build tree). These components may be copied to another folder (such as to another computer).

The folder containing the executables and DLLs should contain the following files:

1. colors.mgk
2. delegates.mgk
3. log.mgk
4. magic.mgk
5. modules.mgk
6. type.mgk
7. type-ghostscript.mgk (if Ghostscript is used)

The "bin" folder should contains all EXE's and DLL's as well as the very important "modules.mgk" file.

With this default setup, you can use any of the command line tools and run scripts as normal. You can actually get by quite nicely this way by doing something like `pushd e:\xxx\yyy\bin` in any scripts you write to execute "out of" this directory.

By default the core of GraphicsMagick on Win32 always looks in the place were the exe program is run from in order to find all of the files as well as the DLL's it needs.

Environment Variables

You can use the "System" control panel to allow you to add and delete what is in any of the environment variables. You can even have user specific environment variables if you wish.

PATH

This sets the default list of places were Windows looks for EXE's and DLL's. Windows CMD shell seems to look in the "current" directory first - no matter what, which may make it unnecessary to update the PATH. If you wish to run any of utilities from another location then you must add the path to your "bin" directory in. For instance, you might add:

```
D:\CVS\GraphicsMagick\VisualMagick\bin
```

to do this for the default build environment like I do.

MAGICK_HOME

If all you do is modify the PATH variable, the first problem you will run into is that GraphicsMagick may not be able to find any of its "modules. Modules are all the IM_MOD*.DLL files you see in the distribution. There is one of these for each and every file format that GraphicsMagick supports. This environment variable tells the system were to look for these DLL's. The compiled in "default" is "execution path" - which says - look in the same place that the application is running "in". If you are running from somewhere other then "bin" - this will no longer work and you must use this variable. If you elect to leave the modules in the same place as the EXE's (a good idea) then you can simply set this to the same place as you did the PATH variable. In my case:

```
D:\\GraphicsMagick\coders
```

This is also the place were GraphicsMagick expects to find the "colors.mgk", "delegates.mgk", "magic.mgk", "modules.mgk", and "type.mgk" files.

One cool thing about the modules build of GraphicsMagick is that you can now leave out file formats and lighten you load. If all you ever need is GIF and JPEG, then simply drop all the other DLL's into the local trash can and get on with your life.

WARNING: Always keep the "xc" format, since IM uses it for internal purposes.

ALSO. You can elect to changes these things the good old "hard-coded" way. Two #defines are applicable.

defines.h has

```
#define MagickConfigurePath  "c:\\GraphicsMagick\\"
```

To view any image in a Microsoft window, type

```
convert image.ext win:
```

Make sure Ghostscript is installed, otherwise, you will be unable to convert or view a Postscript document, and Postscript standard fonts will not be available.

You may use any standard web browser (e.g. Internet Explorer) to browse the GraphicsMagick documentation.

The Win2K executables will work under Windows '98 and later.

Windows Distribution Build Procedure

The following are the instructions for how to build a Q:8 DLL-based distribution installer package using Visual C++ 7.0:

1. Install prerequisite software:
    a. Download and install Inno Setup 5 <"http://www.jrsoftware.org/isinfo.php">.
    b. Download and install ActiveState ActivePerl <"http://www.activestate.com/activeperl/downloads/">.
2. Open workspace VisualMagickconfigureconfigure.sln by double-clicking from Windows Explorer.
    a. Select "Rebuild All"
    b. Click on '!' icon to run configure program
    c. Select DLL build
    d. Check "Build demo and test programs"
    e. Click on Edit "magick_config.h" and ensure that UseInstalledGraphicsMagick and ProvideDllMain are defined.
    f. Finish remaining configure wizard screens to complete.
    g. File -> "Close Workspace"
3. Open workspace VisualMagickVisualDynamicMT.sln by double-clicking from Windows Explorer or opening workpace via Visual C++ dialog.
    a. Build -> "Set Active Configuration" -> "All - Win32 Release" -> OK
    b. Build -> "Rebuild All"
4. Build ImageMagickObject
    a. `cd contrib\win32\ATL7\ImageMagickObject`
    b. `BuildImageMagickObject clean`
    c. `BuildImageMagickObject release`
    d. `cd ..\..\..\..`
5. Open Windows Command Shell Window
    a. `cd PerlMagick`

    b. `nmake clean` (only if this is a rebuild)
    c. `perl Makefile.nt`
    d. `nmake release`

NOTE: access to *nmake* requires that there be a path to it. Depending on how the install of Visual Studio was done, this may not be the case. Visual Studio provides a batch script in VC98Bin called VCVARS32.BAT that can be used to do this manually after you open up a command prompt.

6. Open `VisualMagick\installer\gm-dll-8.iss` by double-clicking from Windows Explorer.
    a. File -> Compile
    b. Test install by clicking on green triangle
7. Test PerlMagick.
    a. `cd PerlMagick`
    b. `nmake test` (All tests must pass!)
8. Test file format read and write.
    a. `cd VisualMagick\tests`
    b. `run_rwfile.bat` (All tests must pass!)
    c. `run_rwblob.bat` (All tests must pass!)
9. Run Magick++ test programs.
    a. `cd Magick++/tests`
    b. `run_tests.bat` (All tests must pass!)
10. Run Magick++ demo programs.
    a. `cd Magick++/demo`
    b. `run_demos.bat`
    c. Use *gmdisplay* to visually inspect all output files.
11. Distribution package is available as `VisualMagick\bin\GraphicsMagick-1.0-Q8-dll.exe`