Compiling Vim on Win32 HOWTO

Dan Sharp <dwsharp at hotmail dot com>

2004-03-02

Revision History

Revision 1.02	2004-03-02	dws
	Updated for changes through 6.2.311	

The goal of this HOWTO is to provide information on compiling Vim 6.2 on Win32 systems with one of the supported compilers: MSVC++, BCC, and GCC (MinGW and Cygwin). If you already have an executable version of Vim that contains the features you want, then you probably don't need this file.

This document is available in HTML, PDF, and plain text formats from
http://vimdoc.sf.net/howto/win32-compile

Table of Contents

1. Introduction

This HOWTO is an attempt to provide information on compiling Vim on Win32 systems. It will walk you through the steps of acquiring the source, getting additional external libraries, and compiling them all into a Vim that suits your editing needs. The general information presented should be applicable to any 6.x version of Vim, though specific patches may be required to enable some specific features, which will be noted when this is the case. This version of the HOWTO is based on features available with the 6.2 version of Vim.

An assumption of this HOWTO is that the required compilers and external interfaces are installed according to the respective program's documentation. While where to get the programs and any special modifications necessary to use them will be noted, general installation information will not.

2. Preparation

In order to compile Vim, you need three things: a compiler, the Vim source code, and utilities to get and extract the source code. Optionally, you may need to install extra programs to enable certain interfaces in Vim (such as Vim's built-in Perl or Python support).

2.1. Getting the Source

Vim's source code is available through three different methods: FTP, CVS, and A-A-P. The simplest method is probably CVS, which requires only a CVS client utility in addition to the compiler.

2.1.1. CVS

The prepatched source code is available via CVS at http://cvs.vim.org . The CVS archive is manually updated by a volunteer as new patches are released, so it may briefly lag behind the currently released patches, but usually no more than a day or two. Check out the code using your preferred CVS client, then simply update your local archive as new patches come out so that you can always have the latest version.

You will only need a CVS client to get your source this way. You can get a command line CVS client as part of the Cygwin tools or from the msys-DTK package with the MinGW tools. Alternatively, you can get WinCVS, a native Win32 GUI application, from www.wincvs.org.

The command to initially checkout the code is:

```
C:\TEMP>cvs -z3 -d :pserver:anonymous@cvs.vim.sourceforge.net:/cvsroot/vim co vim
```

No password is required, so if you are prompted for one, just hit enter. The code will be downloaded from the CVS server and placed in a vim/ subdirectory of your current directory. The -z3 option will compress the files during transit.

Note

If you are unable to access the above site due to firewall / proxy problems, you can try the command

```
C:\TEMP>cvs -z3 -d :pserver:anonymous@cvs-pserver.sf.net:80/cvsroot/vim co vim
```

which will attempt the download via port 80, which most firewalls will allow through, since it is the standard HTTP port.

After the initial checkout, you can keep your archive updated by changing to the `vim` subdirectory and issuing the command:

```
C:\TEMP\Vim>cvs -z3 update -Pd
```

The update command will pull down any files modified since your last checkout / update. The -d flag will cause any new directories created on the CVS server since the last checkout / update to be created locally as well. The -P flag will remove any directories that are made empty by the current update. Similar instructions can be found at http://www.vim.org/cvs.php , including links to other CVS guides and information.

2.1.2. FTP

This manual process requires that you download all the archives, extract them, patch them, and compile them yourself. You need various additional utilities and time. When using this method, it is a good idea to create some batch files to save a lot of retyping and to script tedious commands (like applying the individual patches. Vim 6.1 had 474 of them).

2.1.2.1. Source

The most up-to-date code is available via ftp from ftp.vim.org and its mirrors ftp.xx.vim.org, where xx is replaced with a country code like us, ca, or uk. See http://www.vim.org/mirrors.php for a more complete list of mirror sites. The `/pub/vim/pc` directory has the DOS-formatted source code in zip archives. You will want the `vim62src.zip`, `vim62lang.zip` and either the `vim62rt.zip` or the `vim62rt1.zip` and `vim62rt2.zip` files. Alternatively, you can get the UNIX-formatted, gzipped or bzipped tarballs from `/pub/vim/unix` (either `vim-6.2.tar.bz2` or all four of `vim-6.2-rt1.tar.gz`, `vim-6.2-rt2.tar.gz`, `vim-6.2-src1.tar.gz`, and `vim-6.2-src2.tar.gz`) and `/pub/vim/extra` (`vim-6.2-lang.tar.gz` and `vim-6.2-extra.tar.gz`).

To extract the source from these archives, you will need several utilities. For the zip archives, you will need an unzip program. For the tar archives, you will need a tar program, a gunzip program, and perhaps a bunzip2 program. These utilities are available with the Cygwin distribution or in the MSYS package of the MinGW distribution. Finally, you can get the individual executables from sites like http://unxutils.sf.net or http://gnuwin32.sf.net . If you prefer a GUI program, you can get WinZip (which handles zip, tar, and gzip) from http://www.winzip.com or 7-Zip (which handles zip, tar, gzip, and bzip2) from http://www.7-zip.org.

Whichever source archives you get, place them in the `C:\TEMP` directory together. These archives hold the base 6.2.0 code. Extract the files into the current directory. If you are using the zip archives, you will have a new `vim` subdirectory, and the source will be located in `C:\TEMP\vim\vim62\src`; otherwise, with the tar archives, you will have a `vim62` subdirectory, and the source will be located in `C:\TEMP\vim62\src`.

2.1.2.2. Patches

To get the latest version of Vim, you must apply all the patches from the `/pub/vim/patches` directory for the version of Vim you are compiling. Download all the files starting with 6.2 and place them in the `C:\TEMP` directory with the source archive files. To save some downloading time, every one hundred patches are combined into a single patch file that you can get instead of the one hundred individual patches. If available, they will have filenames like `6.2.1-100` and `6.2.101-200`. You will still have to get the remaining patches individually. For reference, the README file in the `/pub/vim/patches` directory gives a quick synopsis of the bug or feature addressed by each patch, and the header of each patch gives a more complete description of the bug, the way the bug was fixed, and a list of the files affected by the patch.

Some patches address other operating system ports and are not needed by the Win32 version. If you got your source from the tar archives, this is not a problem, because all the source files for all operating systems will be present and get patched, even if they aren't used in the compile. If you got your source from the zip archives, some extra preparation is required.

Files not needed in a Win32 compile are not included in the zip archives. If you try to apply a patch for one of these missing files, you will get a 'File not found' error which can be safely ignored. Some patches, though, patch files in the 'runtime' directory, which doesn't exist in the zip archives. You will get 'File not found' errors for these files as well, but instead of ignoring it, you will want to specify the path to the file. Fortunately, this is as easy as specifying the path it just tried, but leaving off the 'runtime/' part. If you want to avoid retyping a lot of things, you can edit the patch files manually and remove the 'runtime/' from the path. For example, if you have an older version of Vim already installed, you could do:

```
C:\TEMP>vim -c "set lz" -c "bufdo! %s#--- runtime/#--- # | w" -c q 6.2.*
```

to open all the patch files in Vim and loop through each one, removing the 'runtime/' path from the 'to be patched' file specification.

You will need the patch program to apply the patches. Then, simply change to the vim62 directory (of which src and runtime are subdirectories) and issue the command

```
C:\TEMP\vim62>patch -p0 < ../6.2.001
```

or, if you have a more recent version of GNU patch,

```
C:\TEMP\vim62>patch -p0 -i ../6.2.001
```

Note

Some patch programs (like those from UnxUtils and GnuWin32) appear to have problems with unix file formats, which is the format of the patches downloaded from the FTP site. If you get an error like:

```
C:\TEMP\vim62>patch -p0 -i ../6.2.001
patching file src/edit.c
Assertion failed: hunk, file patch.c, line 341

abnormal program termination

C:\TEMP\vim62>
```

trying to apply the patches, run the unix2dos utility against all the patch files to convert them to a DOS file format. The patches should apply cleanly after that. An alternate (and much simpler) approach is to use the --binary option, so that the command line becomes

```
C:\Temp\vim62>patch -p0 --binary -i ../6.2.001
```

This process is then repeated for each patch in order, since some later patches are dependent on earlier patches. To apply all the patches at once, it would be a good idea to combine all the individual patches into a single large patch and apply them all at once. For example:

```
C:\TEMP>copy 6.2.* 6.2-all.patch
C:\TEMP>cd vim62
C:\TEMP\vim62>patch -p0 < ../6.2-all.patch
```

2.1.3. A-A-P

The newest way to get the latest source is with the A-A-P project created by Vim's author, Bram Moolenaar. The project is currently in the alpha stage, so all usual warnings apply. Install the necessary files by follwing the instructions at http://www.a-a-p.org/download.html . We will assume that the program is installed into `C:\Exec`. Since it is a Python application, you will also need a working Python distribution (2.2.x recommended). The program currently runs well with Cygwin's Python, and will run well with ActiveState's Python if you use A-A-P version 0.138 or greater.

Once you have an operational A-A-P installation, download the Vim recipe from http://www.a-a-p.org/vim/main.aap and place it into the directory where you want to build Vim. From a command line in that directory run the aap program with the command

```
C:\TEMP>python C:\Exec\main.py fetch
```

By default, this will extract the source from CVS. Running the program with the command

```
C:\TEMP>python C:\Exec\main.py fetch CVS=no
```

will instead automatically download all the source archives and patches from the FTP site, expand the source, and apply the patches.

As new patches are released, you can run the command again to update your sources from the CVS archive or download and apply them from FTP. Only the new and modified files will be retrieved on the subsequent updates.

More instructions can be found at http://www.a-a-p.org/ports.html Note that running aap with no arguments will download and compile the sources, but it currently uses the Unix configure and Makefile, which won't work for building a Win32 application, so you will only want to use aap with the 'fetch' option for now.

2.2. External Interfaces

Vim can interface with several external libraries to extend its editing functionality. For example, Perl and Python support can be used to enhance Vim's scripting capabilities. The Iconv and Gettext libraries can be used to improve multilingual support. In order to use these features, support for them must be enabled at compile time. Also, the appropriate headers and libraries must be available. Self-installing binary distributions are available for almost all the libraries Vim can use, and installing them will also install the necessary development files Vim requires.

Below is a list of the external libraries which Vim can use, where to get them, and what extra features they provide. Not addressed is how to use the extra features; help on that subject can be found in Vim's help files.

2.2.1. Scripting Interfaces

Vim can work with four external scripting engines to supplement its own built-in scripting language.

2.2.1.1. Perl

Vim can work with Perl versions 5.004 through 5.8. You can download a binary distribution of the latest Perl release from http://www.activestate.com/activeperl and install it on your system. With a Perl-enabled Vim, you can use the :perl and :perldo commands to execute Perl functions. Read :help perl and especially :help perl-using for a description of the Vim's interface with Perl.

2.2.1.2. Python

Vim can work with Python 1.5 through 2.2.2. You can download a binary distribution of the latest Python release from http://www.activestate.com/activepython and install it on your system. With a Python-enabled Vim, you can use the :python and :pyfile commands to execute Python functions. Read :help python and especially :help python-vim for a description of Vim's interface with Python.

2.2.1.3. Ruby

Vim can work with Ruby 1.6.x. You can download a binary distribution of the latest Ruby release from http://rubyinstaller.sourceforge.net and install it on your system. With a Ruby-enabled Vim, you can use the :ruby, :rubyfile, and :rubydo commands to execute Ruby functions. Read :help ruby and especially :help ruby-vim for a description of Vim's interface with Ruby.

2.2.1.4. Tcl

Vim can work with Tcl 8.x. You can download a binary distribution of the latest Tcl release from http://www.activestate.com/activetcl and install it on your system. With a Tcl-enabled Vim, you can use the :tcl, :tclfile, and :tcldo commands to execute Tcl functions. Read :help tcl and especially :help tcl-commands for a description of Vim's interface with Tcl.

2.2.2. Multilanguage Support

Vim can also utilize libraries to improve its support for character encodings and different languages. Technically, you do not need them preinstalled in order to compile Vim, but they must be present in your PATH or in your vim install directory to enable their features in Vim.

2.2.2.1. IConv

The iconv library allows Vim to translate strings (via the iconv() function) from one encoding to another. Vim natively supports translating between UTF-8 and latin1, and among all Windows codepages, but translation to any other encoding will require that support for the iconv library be compiled into Vim. You can get a copy of the iconv library for Win32 from http://sourceforge.net/projects/gettext See :help iconv() for more information.

2.2.2.2. Gettext

Locale and multi-language support can be enabled in Vim with the gettext library. With gettext, messages and menus will appear in the language specified by the user's locale. You can get a copy of the gettext library for Win32 from http://sourceforge.net/projects/gettext See :help multilang for more information.

> **Note**
>
> If you have already installed Vim with the self-extracting installer from ftp://ftp.vim.org/pub/vim/pc/gvim62.exe then you already have libintl.dll in your vim\vim62 install directory and do not need to download it again.

2.2.3. Source Code Browsers

Vim currently supports a couple interfaces that ease traversal of source files.

2.2.3.1. Cscope

New with version 6.2 is Win32 Cscope support, which lets you browse through your source code finding things like all locations a C token is used, what functions call a particular function, and what functions are called by a particular function. See :help cscope and the web sites http://iamphet.nm.ru/cscope/index.html and http://cscope.sf.net/cscope_vim_tutorial.html for more information.

> **Note**
>
> You will need to use the 16.0a binary from http://iamphet.nm.ru/cscope/index.html to use cscope, as the 15.4 binary from http://cscope.sf.net does not appear to work correctly with Vim.

2.2.3.2. SNiFF+

When compiling with Microsoft Visual C++, you can add support for SNiFF+, a source code analysis environment, allowing Vim to display the source files for SNiFF+. See :help sniff for more information.

3. Compiling

Once you have downloaded the source and installed the desired external interfaces, it is time to compile the program. We will assume for the following instructions that you acquired the source via CVS and so the files are found in `C:\TEMP\Vim`.

The file `feature.h` can be edited to match your preferences, though you can skip this to get the default behavior, which should be fine for most people. You can also enable various features by supplying arguments to the make command during the compilation.

All the supported compilers provide a command-line method for compiling Vim. The Vim source contains a makefile in `C:\TEMP\Vim\src` for each compiler. These makefiles provide options specific to each compiler, as well as a common set of options available to all the makefiles. The files are `Make_mvc.mak` and `Make_ivc.mak` (Microsoft Visual C++), `Make_bc5.mak` (Borland C++ 5.x), `Make_ming.mak` (MinGW GCC) and `Make_cyg.mak` (Cygwin GCC). These files have been supplied by George V. Reilly, Ben Singer, Ken Scott and Ron Aaron and have been well tested.

The options for the compiler can be set in three different ways.

1) The simplest is to edit the makefile directly and comment/uncomment the options to build exactly what you want. The problem with this method is that your changes may be lost by updating the files with newer patches.

2) The second method is to define environment variables for the options. For example, to build a GUI and OLE-enabled Vim with the MSVC++ compiler, you could issue the commands:

```
C:\TEMP\VIM\SRC>set GUI=yes
C:\TEMP\VIM\SRC>set OLE=yes
C:\TEMP\VIM\SRC>nmake -f Make_mvc.mak
```

The options would remain in effect for any subsequent builds from the same console window. To change a setting you would change the environment variable.

3) Finally, you can specify the options on the command line itself, like:

```
C:\TEMP\VIM\SRC>make -f Make_cyg.mak GUI=yes OLE=yes
```

This method has the disadvantage that the options must be specified each time, but the advantage that you can see at a glance what options were used for a specific compile. You can avoid retyping all the options for subsequent compiles by using the command history of the console window or putting the command into a script.

Note

If you compile with one set of options and decide to add or remove an option, you will want to do a 'clean' to delete the already-compiled object files. If not, you will probably get errors when compiling or linking the program. Just add 'clean' to the end of the command you used for the compile, and any files generated during the previous compile will be removed. Be sure to include all the same options for the 'clean' as you did for the compile, otherwise you may not get all the right files removed. To clean up after the compile from example three above, use the command:

```
C:\TEMP\VIM\SRC>make -f Make_cyg.mak GUI=yes OLE=yes clean
```

3.1. Commonalities

Attempts have been made to use the same option name for the same feature in each makefile. Enable or disable the feature with a yes or no option. Features common to all the makefiles are:

Option	Description	Default
GUI	'yes' builds a GUI Vim, 'no' builds a console Vim.	yes
OLE	'yes' builds an OLE-enabled Vim.	no
ICONV	'yes' dynamically loads `libiconv.dll` to do character encoding conversions.	yes
GETTEXT	Dynamically load the gettext library `libintl.dll` for multilanguage support.	yes
MBYTE	Enables multibyte support (viewing, not input)	no
IME	Enables IME support	yes
DYNAMIC_IME	Loads the `imm32.dll` library dynamically	yes
CSCOPE	Enables support for the Cscope interface to Vim.	yes
DEBUG	'yes' builds an executable with debug support	no
CPUNR	Specifies the target CPU for the executable. MSVC++, BCC 5.5, and GCC 2.95.x allow i386 through i686, while GCC 3.x.x adds options for pentium2, pentium3, pentium4, athlon, etc. (Check the gcc man page for all the supported targets)	i386
OPTIMIZE	Specifies the optimization level of the executable. Available settings are SPACE (for the smallest executable, SPEED (for a well-balanced executable), and MAXSPEED (for a large but fast executable).	MAXSPEED
POSTSCRIPT	Enables Postscript output for the :hardcopy command	no
FEATURES	Specifies what optional features to use, as given in feature.h of the Vim source. Available options are TINY, SMALL, NORMAL, BIG, and HUGE	BIG
WINVER	Specifies the minimum version of Windows supported by the binary. 0x400 means the binary will run on anything from Win95 on up. 0x500 requires Win2k or higher.	0x400
NETBEANS	Enables the Netbeans integration interface to allow Gvim to be used as the editor for the Netbeans IDE. This option also allows integration with the Agide GUI of the A-A-P project.	yes
XPM	Specifies the location of the XPM headers and library. This option allows Gvim to display XPM graphics with the 'signs' feature.	
PERL	Specifies the directory where Perl is installed.	
PERL_VER	Specifies the version of Perl being used.	56
DYNAMIC_PERL	Indicates whether to statically link the Perl runtime to Vim or to dynamically load it only when needed.	yes
PYTHON	Specifies the directory where Python is installed.	
PYTHON_VER	Specifies the version of Python being used.	22
DYNAMIC_PYTHON	Indicates whether to statically link the Python runtime to Vim or to dynamically load it only when needed.	yes
RUBY	Specifies the directory where Ruby is installed.	
RUBY_VER	Specifies the version of Ruby being used.	16

Option	Description	Default
RUBY_VER_LONG	Specifies the "long" version of Ruby being used.	1.6
DYNAMIC_RUBY	Indicates whether to statically link the Ruby runtime to Vim or to dynamically load it only when needed.	yes
RUBY_PLATFORM	Platform for which this version of Ruby was compiled. The default is for Ruby 1.6. For Ruby 1.8 or higher, the default is i386-mswin32.	i586-mswin32
RUBY_INSTALL_NAME	The name of the main Ruby DLL. For Ruby 1.6, the default value is mswin32-ruby16 and for Ruby 1.8 or higher the default is msvcrt-ruby18.	mswin32-ruby16
TCL	Specifies the directory where Tcl is installed.	
TCL_VER	Specifies the version of Tcl being used.	83
TCL_LONG_VER	Specifies the "long" version of Tcl being used.	8.3
DYNAMIC_TCL	Indicates whether to statically link the Tcl runtime to Vim or to dynamically load it only when needed.	yes

3.2. Compiler Specifics

Vim can be compiled for Win32 systemss using various versions of the Microsoft Visual C++ (versions 4-7), Borland (version 5 and 5.5), or GCC (2.95.x and 3.x.x from Cygwin or MinGW32). Each compiler has its own makefile and its own set of options and features it provides in addition to the ones listed above.

3.2.1. MSVC++

Microsoft's Visual C++ program is a commercial product and is not available for download. You have two options when compiling Vim with MSVC++: command line and IDE.

3.2.1.1. Command line

The most flexible method for compiling Vim from the command line is to use the `Make_mvc.mak` file. The advantage of using this makefile instead of `Make_ivc.mak` is that you have much greater control over the options used to compile Vim. Extra options available are:

Option	Description	Default
GIME	Enables global IME support	no
MAP	Create a mapfile during compilation	yes
SNIFF	Enable the SNiFF+ interface	no
VC6	Delays loading seldom-used DLLs to improve startup time	no

To debug an executable built with `Make_mvc.mak` in the MSDev IDE, you need to use `Make_dvc.mak` in the IDE. From `Make_mvc.mak`:

`Make_mvc.mak` gives a fineness of control which is not supported in Visual C++ configuration files. Therefore, debugging requires a bit of extra work. `Make_dvc.mak` is a Visual C++ project to access that support. To use `Make_dvc.mak`:

1. Build Vim with `Make_mvc.mak`. Use a "DEBUG=yes" argument to build Vim with debug support. E.g. the following builds `gvimd.exe`:

   ```
   C:\TEMP\Vim\src>nmake -f Make_mvc.mak debug=yes gui=yes
   ```

2. Use MS Devstudio and set it up to allow that file to be debugged:

   a. Pass `Make_dvc.mak` to the IDE. Use the File->'Open Workspace' menu entry to load `Make_dvc.mak`. Alternatively, from the command line:

      ```
      C:\TEMP\Vim\src>msdev /nologo Make_dvc.mak
      ```

      Note: `Make_dvc.mak` is in VC4.0 format. Later VC versions see this and offer to convert it to their own format. Accept that. It creates a file called `Make_dvc.dsw` which can then be used for further operations. E.g.

      ```
      C:\TEMP\Vim\src>msdev /nologo Make_dvc.dsw
      ```

      Also, if you got the file from the tar archives, you will need to convert `Make_dvc.mak` with a unix2dos utility before attempting to open it.

   b. Set the built executable for debugging:

      i. Project->Settings (**Alt-F7**) takes you to the Debug dialog.

      ii. Fill "Executable for debug session". e.g. `gvimd.exe`

      iii. Fill "Program arguments". e.g. -R dosinst.c

      iv. Click OK to close the dialog

3. You can now debug the executable you built with `Make_mvc.mak`

Note: `Make_dvc.mak` builds `vimrun.exe`, because it must build something to be a valid makefile.

3.2.1.2. IDE

Vim can be compiled in the MSDev IDE using the `Make_ivc.mak` makefile. From the IDE, go to File->Open Workspace and select `Make_ivc.mak` and click OK. You can then select targets such as Release and/or Debug version of Console Vim or GUI Vim.

Note

`Make_ivc.mak` must be in the DOS file format to successfully be opened by the MSDev IDE. If you get your source from the tar archives or CVS, you will have to convert the file to DOS format using a unix2dos utility. If you have a previous version of Vim installed, then an alternative to the unix2dos utiltity would be the command

```
C:\TEMP>vim -c ":set nomore | bufdo set ff=dos | w" -c q <filenames>
```

Also, a unix2dos batch file that uses gzip to perform the conversion has been supplied by Walter Briscoe and is available from http://vimdoc.sf.net/howto/win32-compile/unix2dos.bat. For those who don't want to download the file, he also suggests the "quick-n-dirty"

```
C:\TEMP>gzip -c %1 | gzip -acd > %2
```

command, which is what his unix2dos.bat does, minus all the error-checking.

You can also use `Make_ivc.mak` to compile from the command line, though you are very limited in the types of executables you can build. Issuing the command

```
C:\TEMP\Vim\src>nmake -f Make_ivc.mak CFG="Vim - Win32 Release gvim OLE"
```

will build a GUI and OLE-enabled Vim. The other available targets are

CFG="Vim - Win32 Debug gvim OLE"
CFG="Vim - Win32 Release gvim"
CFG="Vim - Win32 Debug gvim"
CFG="Vim - Win32 Release vim"
CFG="Vim - Win32 Debug vim"

3.2.2. BCC 5.x

Borland C++Builder 5.0 can be purchased at a local retailer, or the Borland C++ 5.5 command line tools and two service packs are available for download from http://www.borland.com/products/downloads/download_cbuilder.html (While there, you may want to pick up the Borland TurboDebugger). Use the `Make_bc5.mak` makefile to build Vim using either of these compilers. Extra options available are:

Option	Description	Default
BOR	Specifies the installation directory of the BCC Compiler.	C:\BC5
LINK	Specifies the name of the linker	$(BOR)\bin\ilink32
OSTYPE	Build a Win32 executable	WIN32
CODEGUARD	Enables CodeGuard support in the executable	no
USEDLL	Use the Borland runtime dll. Reduces size of the executable, but requires `cc3250.dll` to be in the PATH	no
VIMDLL	Creates a backend `vim32.dll` with a stub frontend executable.	no
ALIGN	Specifies byte alignment	4
FASTCALL	Use register based parameter passing for function calls. Improves performance, but breaks interface to external DLLs, thus is disabled when also compiling with PERL, PYTHON, RUBY, or TCL support.	yes

To build a TCL-enabled vim with DYNAMIC_TCL=yes, you need to download a stub library for version 8.3.x (http://mywebpage.netscape.com/sharppeople/vim/tclstub83-bor.lib) or 8.4.x (http://mywebpage.netscape.com/sharppeople/vim/tclstub84-bor.lib) due to the difference in binary formats between MSVC++ and BCC compiled programs.

Note

If you are using a 5.0x version compiler, you will need to convert the vim.rc file from Unix to DOS format if you did not get it from the zip archives. You can do this with a unix2dos utility as described above in the MSVC++ IDE section.

3.2.3. GCC - MinGW

The MinGW collection can be downloaded from http://www.mingw.org . Use the `Make_ming.mak` makefile to build Vim with the MinGW GCC compiler. A special feature of this makefile is the ability to cross-compile a Win32 executable from Linux. Features include:

Option	Description	Default
ARCH	Specifies the target ARCH for the executable. GCC 2.95.x allows i386 through i686, while GCC 3.x.x adds options for pentium2, pentium3, pentium4, athlon, etc. Check the gcc man page for all the supported targets.	i386
CROSS	Cross-compile from UNIX.	no

Note

NLS support with Mingw (by Eduardo F. Amatria <eferna1 at platea.pntic.mec.es>):

If you want National Language Support, read the file `src/po/README_mingw.txt`. You need to uncomment lines in Make_ming.mak to have NLS defined.

3.2.3.1. Cross-compiling from Linux

(written by Ron Aaron: <ron at mossbayeng.com> with help from Martin Kahlert <martin.kahlert at infineon.com>)

If you like, you can compile the MinGW Win32 version from the comfort of your Linux (or other unix) box. To do this, you need to follow a few steps:

1. Install the MinGW32 cross-compiler (if you have it, go to step 2)

   a. From ftp://ftp.nanotech.wisc.edu/pub/khan/gnu-win32/mingw32/snapshots/gcc-2.95.2-1, get:

   ```
   binutils-19990818-1-src.tar.gz
   mingw-msvcrt-20000203.zip
   gcc-2.95.2-1-x86-win32.diff.gz
   ```

   b. From http://gcc.gnu.org/ get:

   ```
   gcc-2.95.2.tar.gz
   ```

   c. Create a place to put the compiler source and binaries (assuming you are in the home directory):

   ```
   $ mkdir gcc-bin
   $ mkdir gcc-src
   ```

   d. Unpack the sources:

   ```
   $ cd gcc-src
   $ tar xzf ../binutils-19990818-1-src.tar.gz
   $ tar xzf ../gcc-2.95.2.tar.gz
   $ unzip ../mingw-msvcrt-20000203
   ```

   e. Build the different tools:

   ```
   $ export PREFIX=~/gcc-bin/
   $ cd gcc-2.95.2
   $ zcat ../gcc-2.95.2-1-x86-win32.diff.gz | patch -p1 -E
   ```

```
$ cd ../binutils-19990818
$ ./configure --target=i586-pc-mingw32msvc --prefix=$PREFIX
$ make
$ make install
$ cd ../gcc-2.95.2
$ ./configure --target=i586-pc-mingw32msvc \
$ --with-libs=~/gcc-bin/i386-mingw32msvc/lib \
$ --with-headers=~/gcc-bin/i386-mingw32msvc/include \
$ --enable-languages=c++ \
$ --prefix=$PREFIX
$ make
$ make install
```

f. Add `$PREFIX/bin` to your `$PATH`.

2. Get the *unix* version of the vim sources from the tar archives.

3. In `Make_ming.mak`, set `CROSS` to 1 instead of 0.

4. Start your compile:

```
$ make -f Make_ming.mak gvim.exe
```

Now you have created the Windows binary from your Linux box! Have fun...

Note

If you are cross-compiling on Linux with the mingw32 setup and are including the Perl, Python, Ruby, or TCL interfaces, you need to also convert each interpreter's include files to *unix* line-endings. This bash command will do it easily:

```
$ for fil in *.h ; do vim -e -c 'set ff=unix|w|q' $fil
```

3.2.3.2. Building with non-dynamic interpreter support

If for some reason you want to build an executable that does not dynamically load the interpreter DLLs, you need to do a little extra work. Libraries compiled by MSVC++ (like those in the ActiveState distributions) do not link well with GCC. You need to manually create a GCC-compatible library.

The following instruction were originally written by Ron Aaron <ron at mossbayeng.com> for the Python interpreter, but the same instructions apply to other interpreters as well.

This has been tested with the MinGW32 compiler, and the ActiveState ActivePython: http://www.ActiveState.com/Products/ActivePython/ [http://www.ActiveState.com/Products/ActivePython]

After installing the ActivePython, you will have to create a 'MinGW32' `libpython20.a` to link with:

```
$ cd $PYTHON/libs
$ pexports python20.dll > python20.def
$ dlltool -d python20.def -l libpython20.a
```

Once that is done, edit the `Make_ming.mak` so the PYTHON variable points to the root of the Python installation (`C:\Python20`, for example). If you are cross-compiling on Linux with the MinGW32 setup, you need to also convert all the 'Include' files to *unix* line-endings. This bash command will do it easily:

```
$ for fil in *.h ; do vim -e -c 'set ff=unix|w|q' $fil
```

Now just do:

```
$ make -f Make_ming.mak gvim.exe
```

and you will end up with a Python-enabled, Win32 version. Enjoy!

3.2.4. GCC - Cygwin

The Cygwin collection is available from http://www.cygwin.com and provides a Unix-like operating environment under Win32.

3.2.4.1. Win32 vs. Unix

One important item to mention here is that Cygwin can compile two different types of Vim that will run under Windows. One type is a Unix application, the other is a Win32 application. Differences are as follows:

1. The Win32 version is compiled with the Make_cyg.mak makefile, while the Unix version uses the standard `./configure`; `make`;`make install` process.
2. The Win32 version compiles a GUI application that runs like any other native Windows app. The Unix version has GUI support, but it requires an X server (like XFree86) to compile and use it.
3. The Win32 version understands normal DOS paths, like `C:\Windows`. The Unix version understands Unix / Cygwin paths like `/var/log` or `C:/Windows`.

Currently there is no way (and no plans to implement a way) to have a version that runs like the Unix version but sports a native Win32 GUI. With the cygpath.exe utility in the Cygwin distribution and a little work, though, you can help the Win32 version understand Unix / Cygwin paths. Search the tips at http://www.vim.org and the Vim Mailing List Archives at http://groups.yahoo.com/group/vim for information on how to do this.

Note

With the release of XFree86 4.3.0, the Cygwin X server offers two new options, -rootless and -multiwindow. These options allow you to run X programs and have them managed by the native Windows window manager, so they appear like any other standard Windows application.

3.2.4.2. Win32

Use the `Make_cyg.mak` makefile to build Vim with the Cygwin GCC compiler. The initial version of this makefile only supports building a vanilla console Vim (or a vanilla GUI Vim after editing the makefile). Patch 6.1.253 adds many more features to this makefile. In addition to the common ones listed above, it also provides:

Option	Description	Default
ARCH	Specifies the target ARCH for the executable. GCC 2.95.x allows i386 through i686, while GCC 3.x.x adds options for pentium2, pentium3, pentium4, athlon, etc. Check the gcc man page for all the supported targets.	i386

Option	Description	Default
USEDLL	Use the Cygwin runtime DLL. Reduces the size of the executable, but requires that cygwin1.dll be in the PATH.	no

3.3. Additional Goodies

Once you compile your binaries, you are ready to go. In the C:\TEMP\vim\src directory you should have either a `vim.exe` and / or a `gvim.exe`, as well as `install.exe`, `uninstal.exe`, and `vimrun.exe`. There is also `xxd.exe` in `C:\TEMP\vim\src\xxd` and `gvimext.dll` in `C:\TEMP\vim\src\GvimExt`. If you want, you can do a couple more steps to round out your installation.

- Installing `gvimext.dll` will add the "Edit with Vim" set of entries to the context menu when you right-click on files. Prior to version 6.2, you had to manually compile the DLL from the `C:\TEMP\vim\GvimExt` directory. As of 6.2, though, the DLL is compiled as part of the normal build process. To compile it by itself, just go to the `C:\TEMP\vim\src\GVimExt` directory and read the `README.txt` for instructions.
- If you use Visual Studio 6, you may also be interested in the `visvim.dll` extension, which will help integrate Vim into the MSDev IDE (version 6). Go to `C:\TEMP\vim\VisVim` and read the `README.txt` for instructions on compiling (MSVC++ only) and installing the DLL.
- One final step you may want to take is to compress your binaries. The UPX utility (available from http://upx.sf.net will compress each executable to approximately 50% of its original size without creating a noticable penalty in startup time. The MinGW makefile includes a target to run UPX on the compiled Vim or GVim binary. For the others, you can compress all the executable files with the command

  ```
  C:\TEMP\Vim\src>copy xxd\xxd.exe . && upx *.exe
  ```

4. Links

Vim Homepage: http://www.vim.org
Home of this HOWTO: http://vimdoc.sf.net/howto/win32-compile

Utilities:
http://www.a-a-p.org
http://gnuwin32.sf.net
http://unxutils.sf.net
http://www.cygwin.com
http://www.mingw.org
http://www.wincvs.org
http://upx.sf.net

External Interfaces:
http://www.activestate.com/activeperl
http://www.activestate.com/activepython
http://www.activestate.com/activetcl
http://rubyinstaller.sourceforge.net [http://rubyinstaller.sourceforge.net]
http://sourceforge.net/projects/gettext [http://sourceforge.net/projects/gettext]

Compilers:

Cygwin (GCC): http://www.cygwin.com
MinGW (GCC): http://www.mingw.org
BCC 5.5: http://www.borland.com/products/downloads/download_cbuilder.html

5. Appendix A: Pre-Win32 Systems

Vim can be compiled on DOS-based systems as well as Win32 systems. These (usually 16-bit) executables do not support all the options found in the Win32 versions, but they support the basics.

5.1. MS-DOS

Summary:

16 bit, Borland C++ and Turbo C++

```
C:\TEMP\Vim\src>ren Make_bc3.mak Makefile
C:\TEMP\Vim\src>make
```

16 bit, Turbo C

```
C:\TEMP\Vim\src>ren Make_tcc.mak Makefile
C:\TEMP\Vim\src>make
```

32 bit, DJGPP 2.0

```
C:\TEMP\Vim\src>make -f Make_djg.mak
```

32 bit, Borland C++ 5.0 (make sure OSTYPE=DOS16)

```
C:\TEMP\Vim\src>make -f Make_bc5.mak
```

Warning: Be sure to use the right make.exe. Microsoft C make doesn't work; Borland make only works with `Make_bc3.mak`, `Make_bc5.mak` and `Make_tcc.mak`; DJGPP/GNU make must be used for `Make_djg.mak`.

The Borland C++ compiler has been used to generate the MS-DOS executable; it should work without problems. You will probably have to change the paths for `LIBPATH` and `INCLUDEPATH` in the start of the makefile. You will get two warnings which can be ignored (one about _chmod and one about precompiled header files).

The "spawno" library by Ralf Brown was used in order to free memory when Vim starts a shell or other external command. Only about 200 bytes are taken from conventional memory. When recompiling get the spawno library from Simtel, directory `msdos/c`. It is called something like `spwno413.zip`. Or follow the instructions in the Makefile to remove the library.

The Turbo C makefile has not been tested much lately. It is included for those that don't have C++. You may need to make a few changes to get it to work.

DJGPP needs to be installed properly to compile Vim; you need a lot of things before it works. When your setup is OK, Vim should compile with just one warning (about an argument to signal()).

`Make_bc5.mak` is for those that have Borland C++ 5.0 or later. At the top of the file, there are some variables you can change to make either a 32-bit Windows exe (GUI or console mode), or a 16-bit MS-DOS version.

If you get all kinds of strange error messages when compiling, try adding <CR> characters at the end of each line.

5.2. Windows 3.1x

16 bit, Borland C++ 5.0

```
C:\TEMP\Vim\src>make -f Make_w16.mak
```

Warning: Be sure to use the right make.exe. It should be Borland make. Also, the vim16.def file must be in the DOS file format. If you get your sources from CVS or the UNIX tar archives, you will need to convert this file with unix2dos.

You will almost certainly have to change the paths for libs and include files in the makefile. Look for `D:\BC5` and `ctl3dv2`. You will get a number of warnings which can be ignored (_chmod, precompiled header files, and "possibly incorrect assignment").

The makefile should also work for BC++ 4.0 and 4.5, but may need tweaking to remove unsupported compiler & linker options.

5.3. Windows NT with OpenNT

(contributed by Michael A. Benzinger)

Building Vim on OpenNT 2.0 on Windows NT 4.0, with Softway's prerelease gcc:

1. Prepare configure to run the OpenNT shell.

   ```
   $ export CONFIG_SHELL=//D/OpenNT/bin/sh
   ```

2. Make the following exports for modifying config.mk:

   ```
   $ export CFLAGS=-O -Wshadow
   $ export X_PRE_LIBS=-lXmu
   ```

3. Run configure as follows:

   ```
   $ configure --prefix=/vim --bindir=/bin/opennt --enable-gui=Motif
   ```

   If you don't have OpenNTif (Motif support), use this:

   ```
   $ configure --prefix=/vim --bindir=/bin/opennt --enable-gui=Athena
   ```

4. Edit Makefile to perform the following:

   a. Since the Makefile include syntax differs from that of gmake, change #include config.mk to .include "config.mk"

   b. Change all install links to be "ln -f" and not "ln -s".

5. Now build the executable

```
$ make
```