Unofficial MinGW GCC binaries for Windows

UPDATE 2009/10/09: Updated to GCC 4.3.3!

This page contains a friendly installer for GCC 4.3.3 for Windows (native "mingw" win32 version). It's an unofficial release, not promoted by the MinGW project. It features **full Python integration**.

The MinGW maintainers believe that GCC 4.x is not ready for an official release (see for instance this message). I respectfully disagree with them and believe that any release is better than no release, and even more so since GCC is very *hard* to compile. To be clear, let me restate that **these binaries are not endorsed by the MinGW project** in any way. They might or might not work for you (and they happen to work very very well for me).

Also, this installer tries to fix all the problems with compiling Python distutils extensions with GCC (the good ol' MSVCR71.DLL problem, if you know what I am speaking of). See below for details, but it really should Just Works(TM). Using GCC, it is possible to produce extensions (`.pyd`) which are fully compatible with the official python.org distribution (so, they can be mix'n'matched with extensions compiled with Visual Studio).

Download

Download the package here:

- **http://www.develer.com/~rasky/gcc-mingw-4.3.3-setup.exe**

It's GPL licensed (do I really need to say that?)

It contains:

- GCC 4.3.3-tdm-1 (http://www.tdragon.net/recentgcc/)
- binutils 2.19.1
- mingw32-make 3.81-2
- mingwrt 3.16
- w32api 3.13

The source code for all the packages above can be found in many mirrors including the MinGW official download page. Or you can ask them directly to me and I will give you a download link from this site (to be fully GPL compliant *and* save my spare bandwidth).

The source code for the installer (a InnoSetup script) can be found here.

What does the installer do?

- Copies the binary files to run GCC.
- Install the `gccmrt` script (see below).
- Optionally modify the environment PATH.
- Optionally configures an existing Python installation to use GCC for distutils.

Quick guide for installing GCC for Python development

1. Run the installer and follow instructions.

2. If possible, don't select an installation path with whitespaces in it. There is no outstanding bug about this *right now*, but don't abuse your luck.
3. Make sure that the box `Add GCC to your system PATH` is checked.
4. Make sure that the box `Bind to Python installations` is checked.
5. Make sure that the box `Set the default runtime library` is checked. Then, select the library you need:
   - For Python 2.6, 2.7, 3.0 or 3.1, you want `MSVCR90.DLL`
   - For Python 2.4 or 2.5, you want `MSVCR71.DLL`
   - For Python 2.3 or older, you want `MSVCRT.DLL`
6. Mark which Python installations will use GCC by default. It's really up to you 🙂

If you work on multiple Python versions, you will need to switch the runtime library after installation. You can do that at any time with the `gccmrt` script (see below).

FAQ

1. **distutils says:**
   `error: command 'gcc' failed: No such file or directory`.
   One of the following:
   - You didn't let the installer modify your system PATH. Do it yourself manually now and try again.
   - You didn't close your command prompt after installation. You need to close and reopen the command prompt to actually use the new PATH.
2. **Can I work with multiple Python versions and different runtime libraries?**.
   Yes, you can. You just need to call `gccmrt` each time you switch Python version, so that distutils will link against the correct runtime library.
3. **I am scared this installer will screw up the Python installation. How can I rollback, in case something does not work?**.
   Simply run the uninstaller. The uninstaller will absolutely revert any change the installer did, including the Python installation binding (distutils configuration). And you should really relax: all the installer does is to add/modify the `distutils.cfg` in `PYTHON\Lib\distutils`. It's a text file, you can have a look, read the manual, ecc.
4. **Can this GCC be used to compile Python itself from the source code?**.
   No clue about that. Have a try, and use Google. My understanding is that you still need to patch the Python source code somehow.
5. **Why should I use this if I own a Visual Studio .NET 2003, or I'm using the free Visual Studio toolkit?**.
   Because GCC 4.x generates a far better code than Visual Studio .NET 2003 used to do, and has more advanced features like the auto-vectorization. No surprise here, since GCC 4.x is many years newer, after all. You can compare the generated code yourself, but my evidence says so (and pretty clearly so).

Python support (compiling distutils extensions)

This installer provides full support for integration with Python. This basically means 3 things:

- **Add the GCC binary directory to the PATH**.
  Python distutils just tries to run `gcc` and expects to automatically find it (there's no magic discovery involved, like that with Visual Studio where it finds the compiler

through the registry). You don't want to know how many people just forget to set the PATH and waste hours trying to figure out why distutils isn't picking up their GCC (or, worse, it's picking up an older gcc somewhere else, maybe in cygwin...).

- **Set GCC as default compiler in global `distutils.cfg`**.
  The platform-wide `distutils.cfg` (located in `PYTHON\Lib\distutils`) allows to configure the default compiler to use for extension compilation. The installer will show the user all the existing Python installation in the system (discovered through registries) and will let the user choose which ones to configure so that GCC is used by default when compiling extensions.
- **Configure the runtime library to use (`MSVCRT.DLL` or `MSVCR71.DLL`)**.
  As you probably know, Python 2.3 (and older) are compiled with Visual Studio 98 and linked against `MSVCRT.DLL`. Instead, Python 2.4 and 2.5 are compiled with Visual Studio .NET 2003 and linked against `MSVCR71.DLL`. There is some widespread confusion about how to configure GCC properly to handle this. The installer will let the user decide which runtime library to use for compilation. Moreover, the setting can be changed at any time by using the `gccmrt` script (see below).

gccmrt

`gccmrt` is a very simple script which lets you decide which Microsoft runtime library GCC should link against. The choice is permanently "saved", so that every compilation after having run the script will use the specified runtime library. The installer will execute `gccmrt` once at installation time, so you need to manually run it only if you work on different projects using different runtime libraries (eg: different Python versions).

Just by typing `gccmrt` at the prompt (assuming your PATH is configured correctly), you get the usage screen:

```
gccmrt: configure the Microsoft runtime library to use for compilation

Usage: gccmrt <60|70|71|80|90>

  60 - Link with MSVCRT.DLL (like Visual Studio '98 aka VC6)
  70 - Link with MSVCR70.DLL (like Visual Studio .NET aka VC70)
  71 - Link with MSVCR71.DLL (like Visual Studio .NET 2003 aka VC71)
  80 - Link with MSVCR80.DLL (like Visual Studio .NET 2005 aka VC80)
  90 - Link with MSVCR90.DLL (like Visual Studio .NET 2008 aka VC90)
```

I guess you don't need much more instructions...

How gccmrt works (and why it works)

gccmrt works by simply overwriting libmsvcrt.a with the version you requested, and changing the internal "specs" file of GCC to correctly define `__MSVCRT_VERSION__`.

More details to be written... (but trust me, it works!).

Author

I'm Giovanni Bajo. ... for some definition of *author*, since I just wrote the installer script!

DevelerOss: GccWinBinaries (last edited 2011-01-10 18:32:32 by DanieleVarrazzo)