3. Using Python on Windows

This document aims to give an overview of Windows-specific behaviour you should know about when using Python on Microsoft Windows.

3.1. Installing Python

Unlike most Unix systems and services, Windows does not require Python natively and thus does not pre-install a version of Python. However, the CPython team has compiled Windows installers (MSI packages) with every release for many years.

With ongoing development of Python, some platforms that used to be supported earlier are no longer supported (due to the lack of users or developers). Check **PEP 11** for details on all unsupported platforms.

- DOS and Windows 3.x are deprecated since Python 2.0 and code specific to these systems was removed in Python 2.1.
- Up to 2.5, Python was still compatible with Windows 95, 98 and ME (but already raised a deprecation warning on installation). For Python 2.6 (and all following releases), this support was dropped and new releases are just expected to work on the Windows NT family.
- Windows CE is still supported.
- The Cygwin installer offers to install the Python interpreter as well; it is located under "Interpreters." (cf. Cygwin package source, Maintainer releases)

See Python for Windows (and DOS) for detailed information about platforms with precompiled installers.

See also:

Python on XP

"7 Minutes to "Hello World!"" by Richard Dooling, 2006

Installing on Windows

in "Dive into Python: Python from novice to pro" by Mark Pilgrim, 2004, ISBN 1-59059-356-1

For Windows users

in "Installing Python" in "A Byte of Python" by Swaroop C H, 2003

3.2. Alternative bundles

Besides the standard CPython distribution, there are modified packages including additional functionality. The following is a list of popular versions and their key features:

ActivePython
: Installer with multi-platform compatibility, documentation, PyWin32

Enthought Python Distribution
: Popular modules (such as PyWin32) with their respective documentation, tool suite for building extensible Python applications

Notice that these packages are likely to install *older* versions of Python.

3.3. Configuring Python

In order to run Python flawlessly, you might have to change certain environment settings in Windows.

3.3.1. Excursus: Setting environment variables

Windows has a built-in dialog for changing environment variables (following guide applies to XP classical view): Right-click the icon for your machine (usually located on your Desktop and called “My Computer”) and choose *Properties* there. Then, open the **Advanced** tab and click the **Environment Variables** button.

In short, your path is:

> *My Computer* ▸ *Properties* ▸ *Advanced* ▸ *Environment Variables*

In this dialog, you can add or modify User and System variables. To change System variables, you need non-restricted access to your machine (i.e. Administrator rights).

Another way of adding variables to your environment is using the **set** command:

```
set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
```

To make this setting permanent, you could add the corresponding command line to your `autoexec.bat`. **msconfig** is a graphical interface to this file.

Viewing environment variables can also be done more straight-forward: The command prompt will expand strings wrapped into percent signs automatically:

```
echo %PATH%
```

Consult **set /?** for details on this behaviour.

See also:

http://support.microsoft.com/kb/100843
Environment variables in Windows NT

http://support.microsoft.com/kb/310519
How To Manage Environment Variables in Windows XP

http://www.chem.gla.ac.uk/~louis/software/faq/q1.html
Setting Environment variables, Louis J. Farrugia

3.3.2. Finding the Python executable

Besides using the automatically created start menu entry for the Python interpreter, you might want to start Python in the DOS prompt. To make this work, you need to set your **%PATH%** environment variable to include the directory of your Python distribution, delimited by a semicolon from other entries. An example variable could look like this (assuming the first two entries are Windows' default):

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Python25
```

Typing **python** on your command prompt will now fire up the Python interpreter. Thus, you can also execute your scripts with command line options, see *Command line* documentation.

3.3.3. Finding modules

Python usually stores its library (and thereby your site-packages folder) in the installation directory. So, if you had installed Python to `C:\Python\`, the default library would reside in `C:\Python\Lib\` and third-party modules should be stored in `C:\Python\Lib\site-packages\`.

This is how `sys.path` is populated on Windows:

- An empty entry is added at the start, which corresponds to the current directory.
- If the environment variable **PYTHONPATH** exists, as described in *Environment variables*, its entries are added next. Note that on Windows, paths in this variable must be separated by semicolons, to distinguish them from the colon used in drive identifiers (`C:\` etc.).
- Additional "application paths" can be added in the registry as subkeys of `\SOFTWARE\Python\PythonCore\version\PythonPath` under both the `HKEY_CURRENT_USER` and `HKEY_LOCAL_MACHINE` hives. Subkeys which have semicolon-delimited path strings as their default value will cause each path to be added to `sys.path`. (Note that all known installers only use HKLM, so HKCU is typically empty.)

- If the environment variable **PYTHONHOME** is set, it is assumed as “Python Home”. Otherwise, the path of the main Python executable is used to locate a “landmark file” (`Lib\os.py`) to deduce the “Python Home”. If a Python home is found, the relevant sub-directories added to `sys.path` (`Lib`, `plat-win`, etc) are based on that folder. Otherwise, the core Python path is constructed from the PythonPath stored in the registry.
- If the Python Home cannot be located, no **PYTHONPATH** is specified in the environment, and no registry entries can be found, a default path with relative entries is used (e.g. `.\Lib;.\plat-win`, etc).

The end result of all this is:

- When running `python.exe`, or any other .exe in the main Python directory (either an installed version, or directly from the PCbuild directory), the core path is deduced, and the core paths in the registry are ignored. Other “application paths” in the registry are always read.
- When Python is hosted in another .exe (different directory, embedded via COM, etc), the “Python Home” will not be deduced, so the core path from the registry is used. Other “application paths” in the registry are always read.
- If Python can’t find its home and there is no registry (eg, frozen .exe, some very strange installation setup) you get a path with some default, but relative, paths.

3.3.4. Executing scripts

Python scripts (files with the extension `.py`) will be executed by **python.exe** by default. This executable opens a terminal, which stays open even if the program uses a GUI. If you do not want this to happen, use the extension `.pyw` which will cause the script to be executed by **pythonw.exe** by default (both executables are located in the top-level of your Python installation directory). This suppresses the terminal window on startup.

You can also make all `.py` scripts execute with **pythonw.exe**, setting this through the usual facilities, for example (might require administrative rights):

1. Launch a command prompt.
2. Associate the correct file group with `.py` scripts:

```
assoc .py=Python.File
```

3. Redirect all Python files to the new executable:

```
ftype Python.File=C:\Path\to\pythonw.exe "%1" %*
```

3.4. Additional modules

Even though Python aims to be portable among all platforms, there are features that are unique to Windows. A couple of modules, both in the standard library and external, and snippets exist to use these features.

The Windows-specific standard modules are documented in *MS Windows Specific Services*.

3.4.1. PyWin32

The PyWin32 module by Mark Hammond is a collection of modules for advanced Windows-specific support. This includes utilities for:

- Component Object Model (COM)
- Win32 API calls
- Registry
- Event log
- Microsoft Foundation Classes (MFC) user interfaces

PythonWin is a sample MFC application shipped with PyWin32. It is an embeddable IDE with a built-in debugger.

See also:

Win32 How Do I...?
by Tim Golden

Python and COM
by David and Paul Boddie

3.4.2. Py2exe

Py2exe is a `distutils` extension (see *Extending Distutils*) which wraps Python scripts into executable Windows programs (`*.exe` files). When you have done this, you can distribute your application without requiring your users to install Python.

3.4.3. WConio

Since Python's advanced terminal handling layer, `curses`, is restricted to Unix-like systems, there is a library exclusive to Windows as well: Windows Console I/O for Python.

WConio is a wrapper for Turbo-C's `CONIO.H`, used to create text user interfaces.

3.5. Compiling Python on Windows

If you want to compile CPython yourself, first thing you should do is get the source. You can download either the latest release's source or just grab a fresh checkout.

For Microsoft Visual C++, which is the compiler with which official Python releases are built, the source tree contains solutions/project files. View the `readme.txt` in their respective directories:

Directory	MSVC version	Visual Studio version
`PC/VC6/`	6.0	97
`PC/VS7.1/`	7.1	2003
`PC/VS8.0/`	8.0	2005
`PCbuild/`	9.0	2008

Note that not all of these build directories are fully supported. Read the release notes to see which compiler version the official releases for your version are built with.

Check `PC/readme.txt` for general information on the build process.

For extension modules, consult *Building C and C++ Extensions on Windows*.

See also:

Python + Windows + distutils + SWIG + gcc MinGW

or "Creating Python extensions in C/C++ with SWIG and compiling them with MinGW gcc under Windows" or "Installing Python extension with distutils and without Microsoft Visual C++" by Sébastien Sauvage, 2003

MingW – Python extensions

by Trent Apted et al, 2007

3.6. Other resources

See also:

Python Programming On Win32

"Help for Windows Programmers" by Mark Hammond and Andy Robinson, O'Reilly Media, 2000, ISBN 1-56592-621-8

A Python for Windows Tutorial

by Amanda Birmingham, 2004