CHAPTER 2

Installing Pylons

Pylons is written in the Python language and is designed to run on any platform that supports a modern version of Python. It can therefore be used on Windows, Mac OS X, Linux, BSD, and many other platforms. Because Python is an interpreted language, Pylons applications you write for one platform will be able to run on other platforms without any modification.

You can install Pylons in quite a few different ways depending on your needs, but the three main tools most Pylons developers use are as follows:

- A virtual Python environment
- The `easy_install` program
- The Python Package Index

In this chapter, you'll look at what a virtual Python environment is before turning your attention to the Python Package Index and the `easy_install` program. Once you have a thorough understanding of the install processes used by Pylons, you'll turn your attention to Python itself and look at any subtleties you need to be aware of on your particular platform, including how to install packages that include C and C++ extensions.

Note If you don't have a copy of Python installed yet, you might want to jump ahead to the platform-specific notes later in this chapter to learn how to install a recent version of Python such as 2.5 or 2.6 on your platform. However, since almost all platforms apart from Windows already come with a recent version of Python, you'll probably be able to create a virtual Python environment straightaway.

Quick Start to Installation for the Impatient

Pylons is actually very easy to install. If you are not so interested in the details but just want to get up and running with a Pylons installation on Linux as quickly as possible, the following steps show you how. You'll find steps specific to Windows and Mac OS X later in the chapter.

1. Download the `virtualenv.py` script from `http://pylonsbook.com/virtualenv.py`.
2. Create a virtual Python environment in a directory called `env` so that packages you install for Pylons do not affect any other programs using Python on your system:

   ```
   python virtualenv.py --no-site-packages env
   ```

3. Windows users would use `Scripts` instead of bin in the above command but full details are explained later in the Windows-specific instructions. Use the `easy_install` program (which was automatically installed into your virtual Python environment by the previous command) to install Pylons:

   ```
   env/bin/easy_install "Pylons==0.9.7"
   ```

Once the installation has finished, you should always use the programs in `env/bin` rather than the scripts in your system Python installation. For example, where examples in the book specify something like this:

```
$ paster serve --reload development.ini
```

you would actually need to type the following to have the command run from your virtual Python environment:

```
$ env/bin/paster serve --reload development.ini
```

If you don't quite understand the implications of the setup described here, please read the rest of the chapter for the full details.

Installation in Detail

Now that you've seen the commands used to install Pylons, let's take a detailed look at the installation process and what the commands actually do. In the following sections, you'll learn about how packages are stored in a format known as an *egg*, how Easy Install searches online to find the packages you require, and how to install and work with a virtual Python environment.

Using the Python Package Index

You can download all Python packages from a special part of the Python web site known as the Python Package Index (PyPI) at `http://pypi.python.org`. As well as providing the packages, the Python Package Index also contains information about each package such as the author, the project's home page, and a description of the project. Python packages can contain Python source code, configuration files, data files, C or C++ extensions, and metadata about the package.

In Part 3 of the book, you'll learn how you can package up your Pylons applications and automatically upload them to PyPI, but for the moment you need to know only that PyPI is the main place where you can find Python software.

Setting Up a Virtual Python Environment

A *virtual Python environment* is an isolated Python installation set up in such a way that the libraries it contains do not affect programs outside it, making it a good choice for experimenting with new packages or deploying different programs with conflicting library requirements. You can also create a virtual Python environment and install software to it without requiring root or administrator privileges, making it a very useful technique for installing Pylons in a shared environment. *It is highly recommended you install Pylons this way if it's your first time using it.*

A number of tools are available in the Python community for creating a virtual Python environment, but the two most popular are Buildout and `virtualenv.py`. Buildout is popular in the Zope community because it has a lot of features that help you manage all aspects of a deployment, but it can be rather complicated. `virtualenv.py` is a lighter solution designed to handle just the creation of a virtual Python environment, which makes it perfect for most use cases involving Pylons.

To create a virtual Python environment, you need the `virtualenv.py` bootstrapping script. The current version of this script at the time of writing this book is available at `http://pylonsbook.com/virtualenv.py`, and many Linux distributions provide a `python-virtualenv` package, but you will probably want to use the most recent version instead. To obtain the latest version, visit the Python Package Index, and search for `virtualenv`. Download the `.tar.gz` version of the software, and extract the `virtualenv.py` file from the distribution.

You can do so with commands similar to the following, but be sure to update them for the version you want to download:

```
$ wget http://pypi.python.org/packages/source/v/virtualenv/virtualenv-1.1.tar.gz
$ tar zxfv virtualenv-1.1.tar.gz
$ cp  virtualenv-1.1/virtualenv.py ./
```

You can now remove the old files if you like:

```
$ rm -r virtualenv-1.1
$ rm virtualenv-1.1.tar.gz
```

Windows users will need to download the file manually because `wget` is not available, and they will need to use a tool such as 7-zip from `http://www.7-zip.org` to extract the files. Mac users and some Linux users will need to use `curl -O` instead of `wget` to download the file.

■ **Note** If you are using a system that already has the `easy_install` program installed, you could instead install `virtualenv` automatically like this:

```
$ easy_install "virtualenv==1.1"
```

The `virtualenv.py` script will then be available in your Python installation's `bin` or `Scripts` directory.

Whichever way you choose to obtain the `virtualenv.py` script, you can now use it to create an isolated virtual Python environment to keep your Pylons libraries separate from other libraries on your system.

You are now ready to create a virtual Python environment. Here's how it looks on Windows:

```
C:\>C:\Python25\python.exe "C:\Documents and Settings\Administrator\Desktop\virt
ualenv-1.1\virtualenv.py" C:\env
New python executable in C:\env\Scripts\python.exe
Installing setuptools.............................done.
```

Windows users shouldn't create the virtual Python environment in a path with a space. Otherwise, you may see an error similar to this:

```
ValueError: The executable 'C:\\Documents and Settings\\Administrator\\Desktop\\
env\\Scripts\\python.exe' contains a space, and in order to handle this issue
you must have the win32api module installed
```

As the error suggests, installing the `win32api` module from `http://sourceforge.net/project/showfiles.php?group_id=78018` fixes this issue.

On Linux platforms, you should ensure you have the `python-dev` package for your Python version installed; otherwise, the `virtualenv.py` script might complain about include files.

It is also worth noting for advanced users that a virtual Python environment is not necessarily compatible with a customized `distutils.cfg` file.

Working with Easy Install

Now that the virtual Python environment is set up, you can turn your attention to `easy_install`, which is a Python program that automatically fetches packages from PyPI as well as any dependencies it has. It then installs them to the same Python environment where the `easy_install` program is installed.

The `easy_install` program is actually part of a module called `setuptools` and is installed automatically by the `virtualenv.py` script you just ran.

■ **Tip** If you did not use a virtual Python environment, you can still use `easy_install`. Download the `ez_setup.py` file from `http://peak.telecommunity.com/dist/ez_setup.py`, and then run this command:

```
$ python ez_setup.py
```

You can find full documentation for Easy Install at `http://peak.telecommunity.com/DevCenter/EasyInstall`, and although it is a powerful tool with many options for advanced users, its basic use is very straightforward. To give you a flavor of the common ways to use it, I will run through some examples.

To install the latest version of the PasteDeploy package used by Pylons and its dependencies, you would simply run this command:

```
$ easy_install PasteDeploy
```

When you run this command, Easy Install will visit the Python Package Index and the Pylons download page to find the most appropriate version of each of the required packages and install them each in turn. The PasteDeploy package doesn't have any dependencies, but if it did, Easy Install would search the Python Package Index for the most appropriate versions of the dependent packages and automatically download them too.

If you are using a virtual Python environment, you have to add the path to the virtual environment's `bin` directory before the `easy_install` command. If you installed your virtual Python environment to the `env` directory as described earlier, the command would be as follows:

```
$ env/bin/easy_install PasteDeploy
```

On Windows, commands such as `easy_install` are often real Windows applications, so you can add the `.exe` extension to them if you prefer. A virtual Python environment on Windows installs programs to the `Scripts` directory rather than to the `bin` directory, so on Windows the command would be as follows:

```
$ env/Scripts/easy_install.exe PasteDeploy
```

■ **Caution** The rest of the chapters in the book assume you will always add the correct path to your virtual Python environment scripts and the `.exe` extension if it is necessary on your platform.

You can install virtually all packages on the Python Package Index in the same way as you installed the PasteDeploy package here, simply by specifying the package name as the argument to the `easy_install` command.

Installing Pylons

Now that you know how to use `easy_install` on your platform, it is time to install Pylons.

Pylons consists of lots of different packages that all need to be installed for Pylons to work. Pylons itself is distributed under the open source license listed in the preface of the book. All its dependencies are also open source too, but if you are concerned about the details of the licenses, you should check each package.

Rather than leaving you to install each package separately, Pylons uses the Easy Install system you've just learned about to download and install all its dependencies automatically.

At a command prompt, run this command to install Pylons 0.9.7 and its dependencies:

```
$ easy_install "Pylons==0.9.7"
```

At the end of the process, you should have the latest version of Pylons and all its dependencies installed and ready to use.

■ **Tip** If you are a Windows user and are using Python 2.3, you will also need to install a package called `subprocess`, which you can download from `http://www.pylonshq.com/download/subprocess-0.1-20041012.win32-py2.3.exe`. Python 2.4 (and newer) users already have this package.

Incidentally, Pylons may not support Python 2.3 for very much longer because of its lack of decorator support, so you would be wise to upgrade to a more recent version like Python 2.5 or 2.6.

If you want to make sure you have the latest version of Pylons, you can use this command:

```
$ easy_install -U Pylons
```

This will install the latest version if it is not already present or upgrade Pylons to the latest version if an old version has already been installed.

If you are using a virtual Python environment, you will see that the `pylons` module has been installed into the virtual environment:

```
C:\>C:\env\Scripts\python.exe
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pylons
>>>
```

However, if you try to do the same with the system Python executable, you'll get an error because the virtual Python installation has isolated the packages from the main system Python as expected:

```
C:\>C:\Python25\python.exe
Python 2.5.1 (r251:54863, Apr 18 2007, 08:51:08) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pylons
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named pylons
>>>
```

Now that you have successfully installed Pylons and all its dependencies, you are able to move on to Chapter 3, but if you really want a thorough understanding, read on!

Understanding Eggs

Most new Python software including Pylons and all its dependencies are distributed as *eggs*. Eggs, a new package format, have many extra features over the old `distutils` packages, including the addition of dependency information used by the `easy_install` program installed with your virtual Python environment.

■ **Tip** If you haven't come across the egg format before, you can think of eggs as being similar to `.jar` files in Java or `.gem` files in Ruby.

Python eggs are simply ZIP files of Python modules with a few metadata files added, and if you rename them to `.zip`, you can explore their contents. Module ZIP files have been supported in Python since version 2.3, so using a ZIP file instead of a directory structure is nothing new.

In Part 3 of the book, you'll look at how you can create your own egg files from a Pylons application, and I'll discuss more advanced features of eggs such as *entry points*.

Advanced Topics

You've now seen how to set up a virtual Python environment and how to use Easy Install to automatically install Python packages such as Pylons from eggs on the Python Package Index. Before you move on to learning about platform-specific issues, I'll cover a few advanced topics that are worth being aware of to help get the most from your installation.

Activating and Deactivating a Virtual Python Environment

If you are working regularly with the files in a `virtualenv` virtual Python environment, it can become tedious to have to type the `path/to/env/bin/` part in front of every script you want to execute. You can solve this problem by *activating* the environment in which you are currently working. This puts the virtual Python environment executables first in your `PATH` environment variable so that you can run them without typing the full path to the virtual Python environment.

Here's how to do it on Windows:

```
C:\Test>..\env\Scripts\activate.bat
(env) C:\Test>python
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pylons
>>>
```

Once you've activated a virtual Python environment with `activate.bat`, the scripts in the virtual Python environment take precedence over the normal system ones. In the previous example, you can see that entering `python` actually runs the version in your virtual Python environment where Pylons is installed, not the main system Python. The same is true for the other available scripts.

You can deactivate it by typing `deactivate.bat`, and things go back to normal:

```
(env) C:\Test>..\env\Scripts\deactivate.bat
C:\Test>
```

On Linux and Mac OS X, you activate the environment using a Bash script like this:

```
$ source env/bin/activate
```

This script works in Bash and `zsh` and probably most other Bourne-compatible shells. It will not work on `csh` or `tcsh`. If it doesn't work in your non-Bash shell, you'll have to modify it to conform to your shell's syntax. Don't forget the word `source`! That makes it execute in the current shell rather than starting a new process so that it can modify the current shell's `PATH` and other variables. After a shell is activated, notice that the prompt changes to `(env)` to remind you which environment you're in. This is handy if you end up working in multiple virtual environments in different windows.

If you want to deactivate a shell, you would enter this command on Linux and Mac OS X:

```
deactivate
```

The paths will then return to normal, and the prompt will disappear.

Caution There is one potential problem to keep in mind with `activate`/`deactivate`. The shell keeps the path of each command in memory so it doesn't have to look it up again when the command is reexecuted. This bypasses `activate`'s changes to the environment, so `activate` and `deactivate` try to clear this cache, but sometimes they fail. The result is that you type the command name and get the "wrong" one. To make 100 percent sure you're getting the right command, run `which python` (or `which easy_install`, and so on) to see the full path of the command that would have executed. If it's the wrong one, execute it once by typing the full path, and it will update the memory cache to use that version of the command.

If in doubt, it is probably worth using the executables directly without the `activate`/`deactivate` magic.

Setting Virtualenv Options

The `virtualenv.py` script has two interesting command-line options that you can use when creating your virtual Python environment:

`--no-site-packages`: Don't make the global site-packages directory accessible in the virtual environment.

`--clear`: Delete any previous install of this `virtualenv` and start from scratch.

The examples so far have used the `--no-site-packages` option, but you can leave this option out if you prefer your virtual Python environment to also have access to all the packages already installed into the system Python environment.

Choosing Package Versions with Easy Install

Easy Install allows you to be very specific about the version of a particular piece of software you want to install. Let's use the PasteDeploy package again as an example.

To install the latest version of the PasteDeploy package used by Pylons and its dependencies, you would simply run this command, as you've already seen:

```
$ easy_install PasteDeploy
```

To install the PasteDeploy package version 1.7, you would use this:

```
$ easy_install "PasteDeploy==1.7"
```

You can also have more complicated version requirements. For example, to install a version of the PasteDeploy package between 1.5 and 1.7 inclusive, you could do this:

```
$ easy_install "PasteDeploy>=1.5,<=1.7"
```

You can also use Easy Install to upgrade packages. You use the `easy_install` command in a similar way but specify the `-U` flag to tell Easy Install to upgrade the package. Notice how you can still specify version requirements even when upgrading:

```
$ easy_install -U "PasteDeploy>=1.5,<=1.7"
```

Of course, you can also use Easy Install to directly install eggs that exist on your local system. For example, to install the Python 2.5 egg for the PasteDeploy package, you could issue this command:

```
$ easy_install PasteDeploy-1.7-py2.5.egg
```

You can use Easy Install to install source distributions that were created with the old `distutils` module that forms part of the Python distribution and that Easy Install is designed to replace:

```
$ easy_install PasteDeploy-1.5.tar.gz
```

In this case, Easy Install first creates the `.egg` file from the source distribution and then installs it. Easy Install is also equally happy taking URLs as arguments as well as package names or paths to local files on the filesystem.

■ **Tip** Easy Install works by maintaining a file in your Python installation's `lib/site-packages` directory called `easy_install.pth`. Python looks in `.pth` files when it is starting up and adds any module ZIP files or paths specified to its internal `sys.path`. Easy Install simply maintains the `easy_install.pth` file so that Python can find the packages you have installed with Easy Install.

If you ever want to remove a package, the easiest way is to remove its entry from the `easy_install.pth` file. You shouldn't ever need to remove a package, though, because Easy Install will always use the last one you installed by default.

One really useful option for use with `easy_install` is `--always-unzip`. This forces Easy Install to extract all files from the egg packages so that you can browse their source files on the filesystem to see how the packages they contain actually work. That's very handy if you are an inquisitive developer!

Installing with a Proxy Server

If you try to install Pylons using `easy_install` and get a message such as `error: Download error: (10060, 'Operation timed out')`, it might be because your computer is behind an HTTP proxy and so `easy_install` cannot download the files it needs.

For `easy_install` to be able to download the files, you need to tell it where the proxy is. You can do this by setting the `HTTP_PROXY` environment variable. On Linux, you would type this:

```
$ export HTTP_PROXY="http://yourproxy.com:port"
```

or if you need proxy authentication, you would type this:

```
$ export HTTP_PROXY="http://user:password@yourproxy.com:port"
```

On Windows you would set the following:

```
> set HTTP_PROXY=http://your.proxy.com:yourPort
```

You should then be able to run `easy_install Pylons` again, and this time the program will be able to find the files. See the next section for installing Pylons offline if you still have difficulties.

Troubleshooting Easy Install

On rare occasions, the Python Package Index goes down but it is still possible to install Pylons and its dependencies in these circumstances by specifying Pylons' local package directory for installation instead. You can do so like this:

```
$ easy_install -f http://pylonshq.com/download/ Pylons
```

This command tells Easy Install to first check the Pylons web site before going to PyPI. If it finds links to everything it needs on the specified page, it won't have to go to PyPI at all.

■ **Note** If you're using an older version of Pylons, you can get the packages that went with it at the time it was released by specifying the version desired and the Pylons version-specific download site:

```
$ easy_install -f http://pylonshq.com/download/0.9.6.2/ "Pylons==0.9.6.2"
```

You can use the same technique to install any of Pylons' dependencies too. The `-f` option here tells Easy Install to search the URL specified as well as the Python Package Index when looking for packages.

If you can't connect to the Internet at all, you will need to install Pylons offline. Download all of Pylons' dependencies, and place them in a directory called `downloads`. Then use Easy Install to install the software from the directory using this command:

```
$ easy_install -f downloads/ "Pylons==0.9.7"
```

Occasionally Easy Install fails to find a package it is looking for on the Python Package Index. If this situation occurs, you should first ensure you have the latest version of `setuptools` by issuing this command (the `-U` flag means upgrade, as you saw earlier):

```
$ easy_install -U setuptools
```

You will need to do this in Ubuntu 7.04, for example, if you obtained Easy Install via the `python-setuptools*.deb` file rather than as part of your virtual Python environment setup because the version in the `.deb` file is too old for Pylons and its dependencies.

After upgrading, try to install Pylons again, and if one of the dependencies still fails, you will need to manually install that dependency first before attempting the Pylons installation once again.

Another error message you may occasionally encounter when using Easy Install is a `pkg_resources.ExtractionError` error that reads as follows:

```
Can't extract file(s) to egg cache The following error occurred while
trying to extract file(s) to the Python egg cache: [Errno 13] Permission
denied: '/var/www/.python-eggs' The Python egg cache directory is currently set
to: /var/www/.python-eggs Perhaps your account does not have write access to
this directory? You can change the cache directory by setting the
PYTHON_EGG_CACHE environment variable to point to an accessible directory.
```

As the error message suggests, an egg containing a module being imported by `pkg_resources` (another module installed with `setuptools` and `easy_install`) needs to have its contents extracted before the module can be used, but the running script doesn't have permission to extract the egg to the default location. You can change the place the eggs are extracted to by setting the environment variable `PYTHON_EGG_CACHE` to somewhere the application has permission to write to. One way of doing this is as follows:

```
import os
os.environ['PYTHON_EGG_CACHE'] = '/tmp'
```

You would need to add this line before the module import that is failing, and you would usually use a more appropriate location than `/tmp`.

If you are still having problems, you'll need to look online or contact the Pylons mailing list at `pylons-discuss@googlegroups.com`. Two good places to start are the Easy Install documentation at `http://peak.telecommunity.com/DevCenter/EasyInstall` and the TurboGears install troubleshooting guide at `http://docs.turbogears.org/1.0/InstallTroubleshooting`. TurboGears 1.0 uses the same install system as Pylons, so the problems encountered are often similar.

Working on the Bleeding Edge

If you want to use the development version of Pylons—or even contribute to Pylons development—you can install the bleeding-edge latest version directly from the Pylons Mercurial at `http://pylonshq.com/hg`.

Mercurial is a popular open source revision control system written in Python that many projects including Pylons are choosing instead of Subversion because of its distributed nature and more powerful feature set. Mercurial is documented in detail in the Mercurial Book at `http://hgbook.red-bean.com/hgbook.html`, but if you just want to get started quickly, you will find this guide on the Pylons wiki very useful: `http://wiki.pylonshq.com/display/pylonscookbook/Mercurial+for+Subversion+Users`.

Once Mercurial is installed, you will need to clone the development versions of Pylons and all its dependencies. It can be quite a time-consuming process to track down all the repositories and clone the source files, so the Pylons developers created a `go-pylons.py` script to set up a virtual Python environment and automate the process.

Download the program for Pylons 0.9.7 from `http://www.pylonshq.com/download/0.9.7/go-pylons.py`, and then run this command:

```
$ python go-pylons.py --no-site-packages devenv
```

The script whirs away and sets up a development virtual Python environment. The `go-pylons.py` script is being improved all the time and might not always be used only to install development packages. It is likely a future version might also be capable of installing a release version of Pylons.

■ **Caution** Although the Pylons development team always tries to ensure the code in the Pylons trunk is functioning and up-to-date, there is no guarantee it will be stable. If you choose to use the development version instead of the official release, you should be aware that Pylons may not behave as you expect and is more likely to contain some bugs as new features are introduced.

Platform-Specific Notes

Pylons works with all versions of Python since 2.3, but it is recommended that you use Python 2.4 or newer because some of the third-party packages you are likely to use when developing a Pylons application are less likely to support the older versions. Python 2.5 or 2.6 are ideal.

The following sections describe how to install Python on Linux/BSD, Windows, and Mac OS X platforms. They also go into some detail about other tools and software you are likely to need on your platform as well as any extra steps you need to take or platform-specific issues you need to know.

■ **Note** Python supports the use of C or C++ extensions to facilitate integration with other libraries or to speed up certain sections of code. Although you are unlikely to ever need to write your own extensions when developing a Pylons application, you may find some third-party packages, particularly database drivers, do contain extensions.

Unlike pure Python packages, packages containing extensions need to be compiled for each platform on which they are run. Most of the time a binary version of a particular package will already exist for your platform (particularly if you run Windows), but if not, the extension may need to be compiled. The compilation step will happen automatically, but in order for it to work, you will need to set up a suitable development environment. The installation sections for Linux/BSD and Mac OS X platforms will describe how to do this.

Linux and BSD

Most modern Linux and BSD platforms include Python as part of their standard installation. You can find out which version of Python you have on your platform by typing `python` at a prompt and reading the information that is displayed. If the `python` command loads an old version of Python, you might find that the command `python2.5` or `python2.4` loads more recent versions.

If your platform doesn't have a recent version of Python, you will need to install a binary version in whichever way is appropriate for your platform. For example, on Debian you would use the `apt-get` command, on Fedora or Red Hat you would use RPM, and on FreeBSD you would use the packages system.

Compiling Python directly from source is also straightforward, and you're free to do so if you prefer. First download the source distribution from `http://www.python.org/download/source/`, and then extract all the files. One of the extracted files is a `README` file, which includes build instructions for various platforms. You should follow the instructions for your particular platform.

If you want to be able to compile Python packages with C or C++ extensions, you will also need to install a build environment that includes the same version of the GNU Compiler Collection (GCC) as Python and its dependencies were compiled with. Some platforms also require you to install a `python-dev` package as well.

For example, to set up Debian 4.0 to be able to compile Python extensions, you would need to install these development packages:

```
$ sudo apt-get install python-dev libc6-dev
```

Sometimes particular packages need to be compiled with a version of GCC older than the default on the platform. On Debian you can install GCC 2.95 with this command:

```
$ sudo apt-get install gcc-2.95
```

To use this older version, you would set the `CC` environment variable before trying to build a package. The way you do this depends on your shell, but for Bash, you would run this command:

```
$ export CC=/usr/bin/gcc-2.95
```

Once the required version of GCC has been set up, Easy Install should be able to automatically compile any dependencies from the same shell. If you need to open another shell, you will need to check the `CC` variable is still set and set it again if necessary:

```
$ echo $CC
/usr/bin/gcc-2.95
```

Mac OS X

Python 2.5 comes preinstalled on Mac OS X Leopard complete with Easy Install, so Leopard users can get started creating a virtual Python environment and installing Pylons straightaway.

Older versions of Mac OS X also include Python, but the version included is sometimes either one or two years old because of Apple's release cycle. The overwhelming recommendation of the "MacPython" community is for users of old versions of Mac OS X to upgrade Python by downloading and installing a newer version. Visit `http://www.python.org/download/mac/` for more details.

■ **Caution** It is worth being aware that if you install packages to `/Library/Python/2.5/site-packages` and then use `virtualenv` to install Pylons to a local directory, any applications you run from the local directory won't be able to find those in the `/Library/...` path, so it is always best to install everything locally.

If you encounter this problem, you can fix it by running Easy Install from your virtual Python environment for each of the packages that appear to be missing. Easy Install will then find them and add them to `virtual Python environment's easy-install.pth` file.

Windows

If you are using Windows 95, 98, NT, 2000, ME, XP, 2003 Server, or Vista, you can download the Python installer from `http://python.org/download/`. Once you have downloaded the correct version for your platform (which will usually be the x86 version), you simply double-click the installer file and follow the installation instructions (see Figure 2-1).

Figure 2-1. *The Python 2.5 Installer running on Windows XP*

■ **Note** To use the installer, the Windows system must support Microsoft Installer (MSI) 2.0. Just save the installer file to your local machine, and then run it to find out whether your machine supports MSI. Windows XP and newer already have MSI, but many older machines will already have MSI installed too.

If your machine does not have the Microsoft Installer, you can download it:

Windows 95, 98, and ME platforms use this version: `http://www.microsoft.com/downloads/details.aspx?FamilyID=cebbacd8-c094-4255-b702-de3bb768148f&displaylang=en`.

Windows NT 4.0 and 2000 use this version: `http://www.microsoft.com/downloads/details.aspx?FamilyID=4b6140f9-2d36-4977-8fa1-6f8a0f5dca8f&DisplayLang=en`.

On Windows, any scripts that are installed by Easy Install will be put in the `Scripts` directory of your Python installation. By default neither this directory nor your main Python executable are on your `PATH`, so you will not be able to run Python itself or any Python scripts from a command prompt unless you first navigate to the correct directory or specify the full path each time.

To fix this and save yourself a lot of typing in the future, you should add some directories to your `PATH` environment variable. Luckily, this is straightforward to do, so the examples in this book will assume you have set up your path correctly.

Of course, if you are using a virtual Python environment, you should specify the `Scripts` directory within the virtual Python environment rather than the system Python, or you could activate your virtual Python environment instead as described earlier in the chapter.

If you are using Windows 2000 or Windows XP, you can do the following:

1. From the desktop or Start menu, right-click My Computer, and click Properties.
2. In the System Properties window, click the Advanced tab.
3. In the Advanced section, click the Environment Variables button.
4. Finally, in the Environment Variables window, highlight the path variable in the Systems Variable section, and click Edit (see Figure 2-2). Add the text `C:\Python25\;C:\Python25\Scripts` to the end of the path. Each different directory should be separated by a semicolon, so the end of your path might look something like this:

   ```
   C:\Program Files;C:\WINDOWS;C:\WINDOWS\System32➥
   ;C:\Python25\;C:\Python25\Scripts
   ```

5. You might need to restart your computer for the change to take effect.

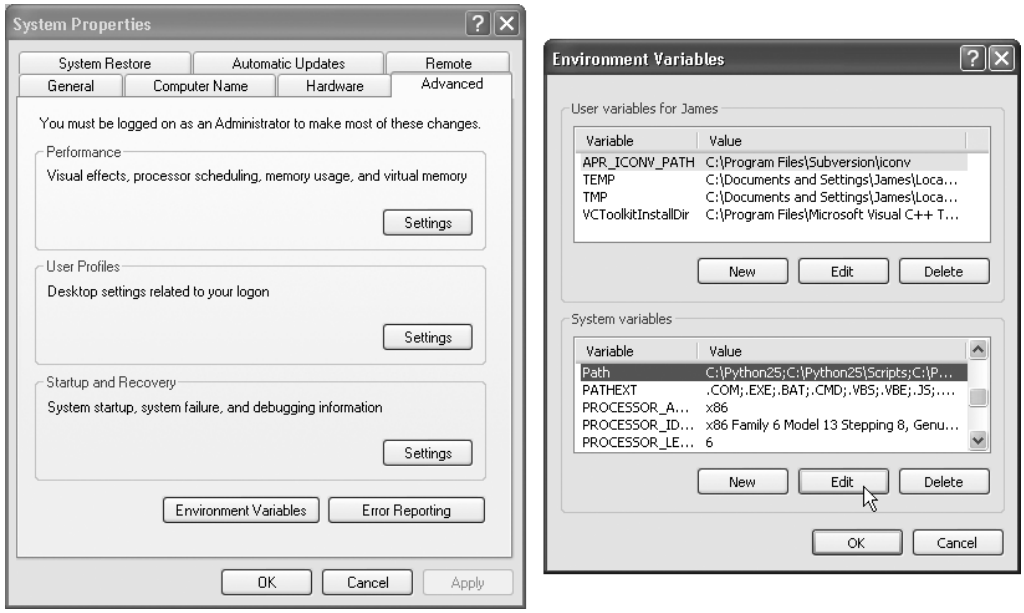

Figure 2-2. *Configuring your* PATH *on Windows XP*

To test your installation has worked and your path is configured correctly, you should select Start ➤ Run and enter the text `cmd` in the input field. When you click OK, a Windows command prompt will load. You can then run Python by typing `python` at the prompt. You should see something similar to what is shown in Figure 2-3.

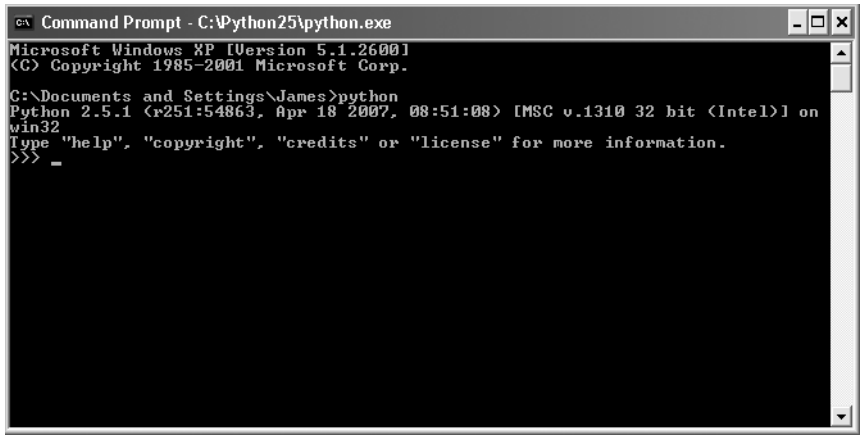

Figure 2-3. *Windows command prompt running Python*

If so, the installation has worked. To exit the Python interactive interpreter, you can press Ctrl+Z followed immediately by pressing the Enter key.

■ **Tip** When you develop Pylons applications, you will find you frequently need access to a Windows command prompt. It can quickly become quite tedious to load a command prompt from the Run option on the Start menu and then manually navigate to the directory containing your Pylons application. To make life easier, Microsoft has released an extension called the Open Command Window Here PowerToy, which allows you to right-click a directory and load a command prompt at that location by choosing Open Command Window Here from the menu. You can download the extension from `http://www.microsoft.com/windowsxp/downloads/powertoys/xppowertoys.mspx`.

There are two slight complications to be aware of when developing Python applications on Windows. The first is that paths on Windows use the \ character as a path separator rather than the / character used on the Linux and Mac OS X platforms. The \ character is treated as an escape character in strings within Python source code, so you cannot use Windows paths in source code strings without first escaping the \ characters. You can do this by adding an extra \ character before each \ in the string. For example, a Windows path might be written like this:

```
my_path = "C:\\Documents and Settings\\James\\Desktop\\Pylons"
```

Luckily, Python also treats / characters in paths on the Windows platform as path separators, so you can also write the same path like this:

```
my_path = "C:/Documents and Settings/James/Desktop/Pylons"
```

■ **Note** Rather than writing different versions of commands for Windows, Linux, and Mac OS X platforms throughout this book, I will instead assume you have set up the `C:\Python25\Scripts` directory to be on your `PATH`. Also, I will write any paths using / characters rather than \ characters, so please be aware that you may have to interpret these slightly differently on Windows.

The second slight complication is the way Windows treats line-end characters. On Unix-like platforms, the newline character `\n` is treated as the line end, whereas on Windows the characters `\r\n` are used. If you have ever loaded a file in Notepad and wondered why it shows an entire paragraph as one long line, it is likely that the file was written on a Unix-like platform and Notepad simply didn't understand the line-end characters.

Luckily, Python understands line-end issues and will work equally well regardless of the line-end characters used, but not all software does. It is generally easiest to stick to one type of line-end character. If you are going to deploy your software on a Unix-like platform, you should strongly consider writing all your Python source files with Unix-style line ends even if you are using Windows. Although FTP software frequently tries to translate Windows line ends to Unix-style line ends, you can save yourself the complication by simply using Unix line ends to start with.

Tip Python comes with a built-in editor called IDLE for editing Python source files that you can read about at `http://www.python.org/idle/doc/idle2.html`. IDLE is a very powerful IDE, but unfortunately it doesn't have an option for choosing which line-end characters to use.

One free editor that does allow you to choose which line ends to use is called SciTE and can be downloaded from `http://scintilla.sourceforge.net/SciTEDownload.html`. You can choose your line-end characters from the menu by selecting Options ➤ Line End Characters. The options are CR+LF, CR, and LF (see Figure 2-4). LF stands for Line Feed and is the Unix line-end character written `\n` in Python strings, and CR stands for carriage return. Windows uses carriage returns and line feeds represented as `\r\n` in Python strings. If you want to convert from one type of line end to another, you can use the Convert Line End Characters option in the Options menu.

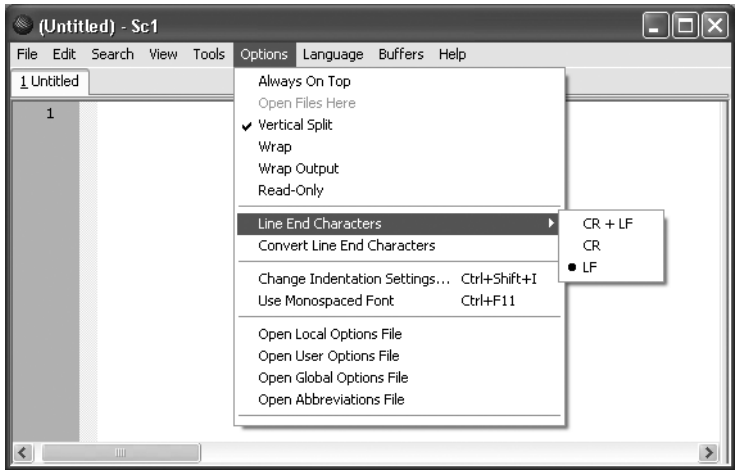

Figure 2-4. *SciTE line endings menu*

Summary

You should now have a very good understanding of all the different tools and techniques used for setting up Python and Pylons, and you should have a virtual Python environment set up and ready to go. With everything in place, it's time to move on to the next chapter and get started learning Pylons.