
Aspen

Release 0.7.1

Chad W. L. Whitacre

March 1, 2007

Zeta Design & Development
<http://www.zetadev.com/software/aspn/>
Email: chad@zetaweb.com

Abstract

Aspen is a web server for highly extensible Python-based publication, application, and hybrid websites.

Introduction

Aspen is designed around the idea that there are basically two kinds of websites, publications and applications, differentiated by their organization and interface models. A *publication* website organizes information into individual pages within a hierarchical folder structure that one navigates by browsing. In an *application* website, on the other hand, data is not organized into hierarchical pages but is dealt with via a non-browsing interface such as a search box.

The HTML version of this documentation is an example of a publication website: a number of hypertext documents organized into sections. If we weren't using LaTeX (or if I knew how to use it better), the sections would probably be encoded in folders. [Gmail](#) is a pure application website, one which organizes and presents information non-hierarchically. Most websites, however, are hybrids. That is, within an overall hierarchical organization you will find both individual pages of information as well as applications such as a site search feature, or a threaded discussion forum.

Publication websites are actually a subset of application websites, of course. An application site can use any interface metaphor; a publication is an application that uses the familiar folder/page metaphor to organize and present its information. Therefore, every website is fundamentally an application.

Aspen enables the full range of websites: publications, applications, and hybrids. It uses the filesystem for the hierarchical structure of publication and hybrid websites, and provides a mechanism for including applications within that hierarchy.

An Aspen website is a collection of files, self-contained within a single directory, called the *root* of the website (cf. [Apache's DocumentRoot directive](#)). In general, URLs map directly to the filesystem. That is, given a root of:

```
/usr/local/www/example.com
```

A request for `/foo.html` would serve a file at:

```
/usr/local/www/example.com/foo.html
```

If all you want to do is serve static files, then that's most of what you need to know.

To extend an Aspen website, you use a UNIX-style userland located within a directory under the website root named `__` (that's two underscores), also called the website's *magic directory*. The existence and contents of this directory are safe from prying eyes, because Aspen will respond to any requests mapping to the magic directory with a [404 Not Found](#).

Installation

Aspen can be installed using either `distutils` or `setuptools`. That is, you can either download a tarball, unpack it, and run:

```
$ python setup.py install
```

Or you can run:

```
$ easy_install aspen
```


Tutorial

Once you have installed Aspen, here are some quick walk-throughs to get your feet wet. They are written sequentially.

3.1 "Greetings, program!"

In your home directory, make a new directory named 'aspentut'. Create a file in 'aspentut' named 'index.html', with the following contents:

```
Greetings, program!
```

At the command line in the 'aspentut' directory, type `aspen`. You should get output like this:

```
$ aspen
aspen starting on ('', 8080)
```

Now open a web browser and hit `http://localhost:8080/`. You should see "Greetings, program!" in your browser. Congratulations!

3.2 Your First Handler

Aspen uses *handlers* to process files such as your 'index.html'. Now we are going to write our own handler.

First, create a directory under 'aspentut' named '__' (that's two underscores). This is 'aspentut's *magic directory*, and it is where you configure and extend your website. Now create two directories under the magic directory: 'etc' and 'lib'. Under 'lib', create a 'python2.x' directory, where 'x' corresponds to the minor version of Python you are using. Your directory structure should now look like this:

```
aspentut
aspentut/__
aspentut/__/etc
aspentut/__/lib/python2.x
```

In '`__lib/python2.x`', create a file named 'handy.py' with the following contents:

```
def handle(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return [environ['PATH_TRANSLATED']]
```

And in `__etc`, create a file named `handlers.conf` with these contents:

```
fnmatch aspen.rules:fnmatch

[handy:handle]
fnmatch *.asp
```

What we have done is we have defined a new handler, and wired it up to be used for any request for a file with the extension `.asp`. So now let's create such a file at `aspentut/handled.asp` and give it the following contents:

```
Greetings, program?
```

Restart Aspen, then hit `http://localhost:8080/handled.asp`. You should see the filesystem pathname of the file being served.

If you are familiar with the WSGI specification, you will recognize that `handy.handle` is a WSGI callable. Aspen plugins all speak WSGI. Also notice that the rules for when a certain handler is invoked are themselves extensible. The `fnmatch` rule comes with Aspen, but you can also write your own.

3.3 What You've Learned

In this brief tutorial we've introduced these key facts about Aspen:

- Aspen websites use the filesystem for site hierarchy.
- Aspen websites are configured and extended via a "magic directory."
- Aspen configuration happens through plain-text configuration files.
- Aspen extensions are WSGI callables.

Besides handlers, Aspen can be extended by wiring up arbitrary WSGI apps to certain paths, and maintaining a global WSGI middleware stack. If this all fits your style of development, then check out the reference documentation that follows for the full story.

Extending Aspen

Aspen uses Python's WSGI specification for its extension architecture. There are three categories of extension:

Category	Explanation
applications	applications are connected to directories within the site hierarchy; only one app touches any given request
handlers	handlers are tied to individual resources (i.e., files and directories) based on extensible rules; only one handler touches
middleware	one or many middleware applications may be specified; all middleware generally touches every request

All extensions are WSGI callables, connected to the above entry points with three configuration files in `'__etc/`:

- `'apps.conf'`
- `'handlers.conf'`
- `'middleware.conf'`

Where called for in these files, objects are specified in a notation derived from `setuptools'` `entry_points` feature: a dotted module name, followed by a colon and a dotted identifier naming an object within the module. This is referred to below as *colon notation*. The following example would import the `bar` object from `example.package.foo`, and use its `baz` attribute (a WSGI callable):

```
example.package.foo:bar.baz
```

The comment character for these files is `#`, and comments can be included in-line. Blank lines are ignored, as is initial and trailing whitespace per-line. Where section names are called for, they are given in brackets.

4.1 Applications: Path-based Extension

In Aspen, an *application* or *app* refers to a WSGI application that is connected to a particular directory. Apps are set up in `'__etc/apps.conf'`.

The `'__etc/apps.conf'` file contains a newline-separated list of white-space-separated path name/object name pairs. The path names refer to URL-space, and are translated literally to the filesystem. If the trailing slash is given, then requests for that directory will first be redirected to the trailing slash before being handed off to the application. If no trailing slash is given, the application will also get requests without the slash. When choosing an application to service a request, the most specific pathname matches first.

Object names are in colon notation, and they name WSGI callables. Aspen updates the `SCRIPT_NAME` and `PATH_INFO` settings in `environ` before handing off to the relevant callable. `SCRIPT_NAME` will never end with a slash, and if `PATH_INFO` is not empty, it will always begin with a slash.

4.3 Middleware: Global Extension

Aspen allows for a full WSGI middleware stack, configured via the ‘`__etc/middleware.conf`’ file. This is simply a newline-separated list of middleware factories in colon notation. Each factory (which may be a class constructor or other callable) is called with exactly one positional argument, the next middleware on the stack. The first-mentioned middleware will therefore be the outer-most in the stack (i.e., closest to the browser).

4.3.1 Example `middleware.conf`

```
example.foo:bar # closest to browser
example.baz:buz # closest to your apps/handlers
```


User Interface (UI)

Users interface with Aspen through three mechanisms: the command line, a configuration file, and the environment. Where a program parameter is set in more than one of these contexts, they take precedence in the order given here. For example, a *mode* option on the command line will override any *mode* setting in the config file or in the environment.

5.1 Command Line

Usage:

```
aspen [options] [command]
```

Aspen takes one optional positional argument, *command*, which must be one of: *start*, *status*, *stop*, *restart*, or *runfg*. The default is *runfg*, which causes Aspen to run in the foreground, sending all messages to stdout.

start, *status*, *stop*, and *restart* control Aspen as a daemon, via a pidfile. If the website root has a directory named ‘__’ (that’s two underscores; the *magic directory*), then the pidfile is at ‘__/var/aspen.pid’. Otherwise, the pidfile is created in ‘/tmp’. When run as a daemon, stdout and stderr are redirected to ‘__/var/aspen.log’ if ‘__’ exists, and to ‘/dev/null’ otherwise. The ‘__/var’ directory will be created if it does not exist. The permission mode of the pidfile is set to 0600; likewise with the logfile, unless it is ‘/dev/null’.

The Aspen distribution includes a script in ‘etc/aspen_bash_completion’ that can be used to configure the bash shell to autocomplete from among Aspen’s arguments. See the source for more information.

Aspen’s command-line options are as follows:

Option	Description
-a/--address=address	The address to which Aspen should bind. If <i>address</i> begins with a dot or a forward slash, then it is interpreted as a relative path.
-m/--mode=mode	One of <i>debugging</i> , <i>development</i> , <i>staging</i> , or <i>production</i> . In <i>debugging</i> and <i>development</i> modes, Aspen will log to stdout.
-r/--root=root	The directory containing the website for Aspen to serve.

See Also:

[mode](#)

Aspen uses the *mode* module to model the application life-cycle. It is available to your applications at `aspen.mode`

5.2 Configuration File

This section describes the general Aspen configuration file at ‘`__/etc/aspen.conf`’. Additional configuration files are described in the “Extending Aspen” chapter. ‘`aspen.conf`’ is in ‘.ini’-style format per the `ConfigParser` module. Aspen responds to the following settings in the `main` section. You may define additional settings and sections that are meaningful to your application, which you may access using the `aspen.conf` object described below in the “API” chapter.

Option	Description
<i>address</i>	The address to which Aspen should bind. If <i>address</i> begins with a dot or a forward slash, then it is interpreted as an <code>ADDRESS</code> .
<i>defaults</i>	A comma-separated list of names to look for when a directory is requested. Any default resource is located immediately.
<i>http_version</i>	The version of HTTP to speak, either <code>1.0</code> or <code>1.1</code> .
<i>mode</i>	One of <code>debugging</code> , <code>development</code> , <code>staging</code> , or <code>production</code> . In <code>debugging</code> and <code>development</code> modes, Aspen will restart itself any time the following configuration files or any module source files change on the filesystem:
<i>threads</i>	The number of threads to maintain in the request-servicing thread pool.

5.2.1 Example

Here is an example ‘`aspen.conf`’ configuration file:

```
[main]
address = :8000

[myapp]
knob = true
```

See Also:

[Extending Aspen](#)

Aspen’s three additional configuration files are described here.

[mode](#)

Aspen relies on this module to model the application life-cycle. It is available to your applications at `aspen.mode`.

5.3 The Environment

Aspen incorporates the `mode` module, which uses the `PYTHONMODE` environment variable to model the application life-cycle through four deployment modes: `debugging`, `development`, `staging`, and `production`. This module is available to your applications at `aspen.mode`, and its API is documented in the “API” chapter, below.

Aspen itself adapts to the current `PYTHONMODE`. In `debugging` and `development` modes, Aspen will restart itself any time the following configuration files or any module source files change on the filesystem:

- ‘`apps.conf`’
- ‘`aspen.conf`’
- ‘`handlers.conf`’
- ‘`middleware.conf`’

See Also:

`mode`

Aspen relies on this module to model the application life-cycle. It is available to your applications at `aspen.mode`

Application Programming Interface (API)

Aspen parses and harmonizes all command-line, configuration file, and environment settings before it loads your plugins. This information is then available to your modules via several objects which are dynamically placed in the global `aspen` namespace before your plugins are loaded—`conf`, `configuration`, and `paths`—and via the `mode` module.

6.1 The `aspen.conf` object

The `aspen.conf` object is an instance of `aspen._configuration.ConfFile`, which subclasses the standard library's `ConfigParser.RawConfigParser` class to represent the `'__/etc/aspen.conf'` file. In addition to the `RawConfigParser` API, the object supports both attribute and key read-only access; either returns a dictionary corresponding to a section of the `'aspen.conf'` file. If the named section does not exist, an empty dictionary is returned.

Your application is free and encouraged to use the `'aspen.conf'` file for its own configuration, and to access that information via this object.

To illustrate, here is a minimal `'aspen.conf'` file:

```
[my_settings]
foo = bar
```

Such a file could support code like this:

```
import aspen

def wsgi_app(envIRON, start_response):
    my_setting = aspen.conf.my_settings.get('foo', 'default')
    start_response('200 OK', [])
    return ["My setting is %s" % my_setting]
```

See Also:

[RawConfigParser](#)

In addition to the API above, `aspen.conf` also exposes the `RawConfigParser` API.

6.2 The `aspen.configuration` object

The `aspen.configuration` object provides raw access to the parser objects used to configure your server, and a number of basic settings.

6.2.1 Parsers

The various parsers and raw settings are exposed as these members:

args

An argument list as returned by `optparse.OptionParser.parse_args`.

conf

An instance of `aspen._configuration.ConfFile`; see below.

optparser

An `optparse.OptionParser` instance.

opts

An `optparse.Values` instance per `optparse.OptionParser.parse_args`.

paths

An instance of `aspen._configuration.Paths`; see below.

6.2.2 Settings

Furthermore, `aspen.configuration` exposes specific configuration settings as these members:

address

A *(hostname, port)* tuple (for `AF_INET` and `AF_INET6` address) or string (for `AF_UNIX`) giving the address to which Aspen is bound.

command

A string giving the command line argument (*start, stop, etc.*).

daemon

A boolean indicating whether Aspen is acting as a daemon.

defaults

A tuple listing the default resource names to look for in a directory.

sockfam

One of `socket.AF_INET`, `socket.AF_INET6`, and `socket.AF_UNIX`.

threads

A non-zero positive integer; the number of threads in the server's request-handling thread pool.

All members are intended to be read-only.

See Also:

[ConfigParser](#)

The naming is not PEP 8, but the documentation is fine.

[optparse](#)

On the other hand, the documentation for `optparse` is rather, um, convoluted. Good luck!

6.3 The `aspen.mode` module

It is often valuable to maintain a distinction between various phases of an application's lifecycle. The `mode` module calls these phases *modes*, and identifies four of them, given here in conceptual life-cycle order:

Mode	Description
<code>debugging</code>	The application is being actively debugged; exceptions may trigger an interactive debugger.
<code>development</code>	The application is being actively developed; however, exceptions should not trigger interactive debugging.
<code>staging</code>	The application is deployed in a mock-production environment.
<code>production</code>	The application is in live use by its end users.

The expectation is that various aspects of the application—logging, exception handling, data sourcing—will adapt to the current mode. The mode is set in the `PYTHONMODE` environment variable. This module provides API for interacting with this variable. If `PYTHONMODE` is unset, it will be set to `development` when this module is imported.

6.3.1 Members

The module defines the following functions:

`get()`

Return the current `PYTHONMODE` setting as a lowercase string; will raise `EnvironmentError` if the (case-insensitive) setting is not one of `debugging`, `development`, `staging`, or `production`.

`set(mode)`

Given a mode, set the `PYTHONMODE` environment variable and refresh the module's boolean members. If given a bad mode, `ValueError` is raised.

`setAPI()`

Refresh the module's boolean members. Call this if you ever change `PYTHONPATH` directly in the `os.environ` mapping.

The module also defines a number of boolean attributes reflecting the current mode setting, including abbreviations and combinations. Uppercase versions of each of the following are also defined (e.g., `DEBUGGING`).

`debugging, deb`

True if `PYTHONMODE` is set to `debugging`.

`development, dev`

True if `PYTHONMODE` is set to `development`.

`staging, st`

True if `PYTHONMODE` is set to `staging`.

`production, prod`

True if `PYTHONMODE` is set to `production`.

`debugging_or_development, debdev, devdeb`

True if `PYTHONMODE` is set to `debugging` or `development`.

`staging_or_production, stprod`

True if `PYTHONMODE` is set to `staging` or `production`.

6.3.2 Example

Example usage:

```

>>> import mode
>>> mode.set('development')      # can set the mode at runtime
>>> mode.get()                   # and access the current mode
'development'
>>> mode.development             # module defines boolean constants
True
>>> mode.PRODUCTION             # uppercase versions are also defined
False
>>> mode.dev                     # as are abbreviations
True
>>> mode.DEBDEV, mode.stprod     # and combinations
(True, False)

```

See Also:

[mode](#)

The mode module is maintained as part of the `lib537` library.

6.4 The `aspen.paths` object

The `aspen.paths` object is an instance of `aspen._configuration.Paths`; it is simply a container for various paths, all normalized and absolute:

root

the website's filesystem root

the magic directory

lib

the site's local Python library, `'__lib/python2.x'`

pkg

'site-packages' under the site's local Python library, `'__lib/python2.x'`

plat

the local platform-specific Python library, `'__lib/plat-x'`

If there is no magic directory, then `---`, `lib`, `pkg` and `plat` are all `None`. If there is, then `lib`, `pkg` and `plat` are added to `sys.path`.

Credits and Legalese

All original work is copyright (c) 2006-2007 Chad Whitacre and contributors, all rights reserved, and is released under the [MIT license](#):

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.1 aspen_bash_completion

The 'aspen_bash_completion' script is based on a similar script from Django. The original is copyright (c) 2005 by the Lawrence Journal-World, all rights reserved, and is used in modified form under the BSD license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of Django nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7.2 daemon.py

The 'daemon.py' module is copyright 2006 by LivingLogic AG, Bayreuth/Germany and Walter Drwald. It is used without change under the following license:

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of LivingLogic AG or the author not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

LIVINGLOGIC AG AND THE AUTHOR DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL LIVINGLOGIC AG OR THE AUTHOR BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

7.3 wsgiserver.py

The 'wsgiserver.py' module is copyright (c) 2004-2006 by the CherryPy Team (team@cherrypy.org), all rights reserved, and is used without change under this BSD license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the CherryPy Team nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.