
Aspen

Release 0.3

Chad W. L. Whitacre

November 17, 2006

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 http://aspen.googlecode.com/svn/tags/0.3/
```


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 A Python Script

Aspen supports pseudo-CGI-style programming with more or less regular python scripts. In your 'aspentut' directory, create a file named 'foo.py' with the following contents:

```
response.body = "Greetings, program!"
response.headers['Content-Type'] = 'text/plain'
```

Assuming that Aspen is still running, hit `http://localhost:8080/foo.py` in your browser. You should again see "Greetings, program!"

3.3 Your First Handler

Aspen uses *handlers* to process your 'index.html' and 'foo.py' files. 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):
    return environ['aspen.fp'].name
```

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. That's because `environ['aspen.fp']` points to an open file object of that file.

If you are familiar with the WSGI specification, you will recognize that `handy.handle` is very nearly a WSGI callable. Aspen plugins all speak a slight superset of 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.

3.4 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 slightly extended 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) based on extensible rules; only one handler touches any given request |
| 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
```

Wherever in these files a name in colon notation points to a class, that class is instantiated with the current `Website` instance as its sole positional argument, and the instance is used as the named object.

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 will (over)write a file called 'README.aspen' in each directory mentioned in 'apps.conf', containing the relevant line from 'apps.conf'. If the directory does not exist, it is created.

4.1.1 Example apps.conf

```
/foo          example.apps:foo    # will get both /foo and /foo/  
/bar/         example.apps:bar    # /bar will redirect to /bar/  
/bar/baz      example.apps:baz    # will 'steal' some of /bar's requests
```

4.2 Handlers: Resource-based Extension

Aspen *handlers* are WSGI applications that are associated with files on the filesystem according to arbitrary rules. This enables ASP/PHP-style web development, where URLs map literally to the filesystem, and the response is generated by somehow processing a filesystem resource.

The '___etc/handlers.conf' file begins with an anonymous "rules" section, which is a newline-separated list of white-space-separated rule name/object name pairs. Rule names can be any string without whitespace. Each object name (in colon notation) specifies a *rule*, a callable taking a Python file object and an arbitrary predicate string, and returning True or False.

Following the rule specification are sections specifying *handlers*, which as mentioned above are WSGI callables. When called, handlers receive the following additional objects in `environ`:

| key | value |
|----------------------|---|
| <i>aspen.fp</i> | the filesystem resource as a <code>file</code> object, positioned at zero |
| <i>aspen.website</i> | the <code>Website</code> instance |

The name of each section specifies a handler (a WSGI callable) in colon notation. The body of each section is a newline-separated list of conditions under which this handler is to be called. Fundamentally, these conditions are made up of a rule name as defined at the beginning of the file, and an arbitrary predicate string (which can include whitespace) that is meaningful to the matching rule callable. If no predicate is given, then the rule callable will receive `None` for its predicate argument. Rules must be explicitly specified at the beginning of the file before being available within handler sections. After the first condition in a handler section, additional condition lines must begin with one of `AND`, `OR`, or `NOT`. These case-insensitive tokens specify how conditions are to be combined in evaluating whether to use this handler.

On each request, handlers are considered in the order given, and the first matching handler is used. Only one handler is used for any given request.

Note that if the file '___etc/handlers.conf' exists at all, the defaults (see the example below) disappear, and you must respecify any of the default rules in your own file if you want them.

4.2.1 Example handlers.conf

This is Aspen's default handler configuration:

```

fnmatch      aspen.rules:fnmatch
hashbang     aspen.rules:hashbang
mime-type    aspen.rules:mimetype

[aspen.handlers:HTTP404]
fnmatch *.py[cod]          # hide any compiled Python scripts

[aspen.handlers:pyscript]
    fnmatch      *.py          # exec python scripts ...
OR hashbang          # ... and anything starting with #!

[aspen.handlers:Simplate]
mime-type text/html        # run html files through the Simplates engine

[aspen.handlers:static]
fnmatch *                  # anything else, serve it statically

```

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 specifiers in colon notation. The first-mentioned middleware will be the outermost in the stack (i.e., closest to the browser).

Aspen adds an `http.py.Responder` instance to the bottom of the stack.

4.3.1 Example `middleware.conf`

```

example.foo:bar # called as is
example.baz:Buz # will be instantiated

```