pyramid_assetmutator Documentation

Release 0.1

Seth Davis

September 16, 2016

Contents

CHAPTER 1

Overview

`pyramid_assetmutator` provides simple and flexible asset mutation (also known as compiling or piping) for your Pyramid applications.

Inspired by other more powerful asset management packages, its goal is to provide a basic and straightforward mechanism for asset *compilation* (e.g. CoffeeScript/LESS), *minification* (e.g. jsmin), and *optimization* (e.g. pngcrush).

Warning: This package only supports Pyramid 1.3 or later.

CHAPTER 2

Installation

To install, simply:

```
pip install pyramid_assetmutator
```

- You'll need to have Python 2.6+ and pip installed.

CHAPTER 3

Setup

Once `pyramid_assetmutator` is installed, you must include it in your Pyramid project's configuration. This is typically done using Pyramid's `config.include` mechanism in your project's `__init__.py`:

```
config = Configurator(...)
config.include('pyramid_assetmutator')
```

Next, you must assign one or more *mutators* via the new *`assign_assetmutator()`* configuration method, so that it knows what type of assets you want mutated. The configuration syntax for your Pyramid project's `__init__.py` is:

```
config.assign_assetmutator('CURRENT EXTENSION', 'COMMAND', 'NEW EXTENSION')
```

For example, the following configuration would activate `pyramid_assetmutator` and initialize mutators for CoffeeScript and LESS files (allowing them to be compiled into the appropriate JavaScript and CSS):

```
config = Configurator(...)
config.include('pyramid_assetmutator')
config.assign_assetmutator('coffee', 'coffee -c -p', 'js')
config.assign_assetmutator('less', 'lessc', 'css')
```

CHAPTER 4

Usage

Once you have configured your mutators, you can use one of the provided view helper methods in your templates to reference (with Pyramid's asset specification syntax) and "mutate" (if needed) an asset.

Four view helper methods are provided depending on your desired results:

assetmutator_url(*path*, ***kw*)
Returns a Pyramid `static_url()` of the mutated asset (and mutates the asset if needed).

Parameters

- **path** (*string - Required*) – The Pyramid asset path to process.
- **mutator** (*dict or string - Optional*) – Allows you to override/specify a specific mutator to use (e.g. `coffee`), or assign a brand new mutator dictionary to be used (e.g. `{'cmd': 'lessc', 'ext': 'css'}`)

assetmutator_path(*path*, ***kw*)
Returns a Pyramid `static_path()` of the mutated asset (and mutates the asset if needed).

Parameters

- **path** (*string - Required*) – The Pyramid asset path to process.
- **mutator** (*dict or string - Optional*) – Allows you to override/specify a specific mutator to use (e.g. `coffee`), or assign a brand new mutator dictionary to be used (e.g. `{'cmd': 'lessc', 'ext': 'css'}`)

assetmutator_source(*path*, ***kw*)
Returns the source data/contents of the mutated asset (and mutates the asset if needed). This is useful when you want to output inline data (e.g. for inline JavaScript blocks).

Parameters

- **path** (*string - Required*) – The Pyramid asset path to process.
- **mutator** (*dict or string - Optional*) – Allows you to override/specify a specific mutator to use (e.g. `coffee`), or assign a brand new mutator dictionary to be used (e.g. `{'cmd': 'lessc', 'ext': 'css'}`)

Note: Many template packages escape output by default. Consult your template language's syntax to output an unescaped string.

---

assetmutator_assetpath(*path*, ***kw*)
Returns a Pyramid asset specification such as `pkg:static/path/to/file.ext` (and mutates the asset if needed).

Parameters

- **path** (*string – Required*) – The Pyramid asset path to process.
- **mutator** (*dict or string – Optional*) – Allows you to override/specify a specific mutator to use (e.g. `coffee`), or assign a brand new mutator dictionary to be used (e.g. `{'cmd': 'lessc', 'ext': 'css'}`)

This function could be used to nest `pyramid_assetmutator` calls. e.g. `assetmutator_path(assetmutator_assetpath('pkg:static/js/script.coffee'))` could compile a CoffeeScript file into JS, and then further minify the JS file if your mutator configuration looked something like:

```
config.assign_assetmutator('coffee', 'coffee -c -p', 'js')
config.assign_assetmutator('js', 'uglifyjs', 'js')
```

4.1 Examples

An example using the Chameleon template language (and assuming that a mutator has been assigned for "coffee" files):

```
<script src="${assetmutator_url('pkg:static/js/test.coffee')}"
        type="text/javascript"></script>
```

And now the same example, but for `inline` code output:

```
<script type="text/javascript">
${structure: assetmutator_source('pkg:static/js/test.coffee')}
</script>
```

Or, if your default JS mutator configuration uses `jsmin`, but you wanted to use `uglifyjs` for a particular asset:

```
<script src="${assetmutator_url('pkg:static/js/test.js', mutator={'cmd': 'uglifyjs', 'ext': 'js'})}"
        type="text/javascript"></script>
```

Lastly, *`assetmutator_assetpath()`* is a particularly nifty/dirty method which gives you the ability to chain mutators. For example, if you wanted to mutate a CoffeeScript file into a JavaScript file and then minify the JavaScript file, you could do something like:

```
<script src="${assetmutator_url(assetmutator_assetpath('pkg:static/js/test.coffee'))}"
        type="text/javascript"></script>
```

CHAPTER 5

Mutators

You can assign as many mutators as you like using the `config.assign_assetmutator` method, but it is important to keep in mind the following:

- The mutator `COMMAND` must be installed, must be executable by the Pyramid process, and by default must *output the mutated data to stdout*. The last point can get tricky depending on the command, so be sure to check its command switches for the appropriate option (or create a wrapper as seen below).
- Mutators are executed in order (first in, first out), which means that it is possible to compile a CoffeeScript file into a JavaScript file and then minify the JavaScript file; but for certain configuations this may only work if you have assigned the CoffeeScript compiler before the JavaScript minifier.

Here are a few mutator commands that have been tested and are known to work as of this writing:

```
# CoffeeScript - http://coffeescript.org/
config.assign_assetmutator('coffee', 'coffee -c -p', 'js')

# Dart - http://www.dartlang.org/
# Requires a wrapper - http://gist.github.com/98aa5e3f3d183d908caa
config.assign_assetmutator('dart', 'dart_wrapper', 'js')

# TypeScript - http://www.typescriptlang.org/
# Requires a wrapper - http://gist.github.com/eaace8a89881c8ca9cda
config.assign_assetmutator('ts', 'tsc_wrapper', 'js')

# LESS - http://lesscss.org/
config.assign_assetmutator('less', 'lessc', 'css')

# SASS/SCSS - http://sass-lang.com/
config.assign_assetmutator('sass', 'sass', 'css')
config.assign_assetmutator('scss', 'sass --scss', 'css')

# jsmin - http://www.crockford.com/javascript/jsmin.html
config.assign_assetmutator('js', 'jsmin', 'js')

# UglifyJS - http://github.com/mishoo/UglifyJS
config.assign_assetmutator('js', 'uglifyjs', 'js')

# pngcrush - http://pmt.sourceforge.net/pngcrush/
# Requires a wrapper - http://gist.github.com/3a0c72ef9bb217315347
config.assign_assetmutator('png', 'pngcrush_wrapper', 'png')
```

CHAPTER 6

Settings

Additional settings are configurable via your Pyramid application's `.ini` file (in the app section representing your Pyramid app) using the `assetmutator` key:

`assetmutator.remutate_check` *Default: mtime*

Specifies what type of method to use for checking to see if an asset source has been updated and should be re-mutated. If set to `exists` (fastest, but not usually ideal), then it will only check to see if a filename matching the mutated version of the asset already exists. If set to `mtime`, then only the last modified time will be checked. If set to `checksum` (slowest, but most reliable), then the file contents will be checked.

`assetmutator.asset_prefix` *Default: _*

A prefix to add to the mutated asset filename.

`assetmutator.mutated_path` *Default: [none]*

By default, mutated files are stored in the same directory as their source files. If you would like to have all mutated files stored in a specific directory, you can define a Pyramid asset specification here (e.g. `pkg:static/cache/`).

Note: The specified path must be a valid asset specification that matches a configured static view, and must be writable by the application.

`assetmutator.each_request` *Default: true*

Whether or not assets should be checked/mutated during each request when the template language encounters one of the `assetmutator_*` methods.

`assetmutator.each_boot` *Default: false*

Whether or not assets should be checked/mutated when the application boots (uses Pyramid's `ApplicationCreated` event).

Note: If set to true, then you must specify the `asset_paths` to be checked (see below).

`assetmutator.asset_paths` *Default: [none]*

Which path(s) should be checked/mutated when the application boots (only loaded if `assetmutator.each_boot` is set to true).

> **Note:** Asset path checks are not recursive, so you must explicitly specify each path that you want checked.

For example, if you wanted to only check/mutate assets on each boot (a good practice for production environments), and only wanted to process the `js` and `css` directories, and would like each mutated `_filename` to be saved in a `myapp:static/cache/` directory, then your `.ini` file could look something like:

```
[app:main]
...other settings...
assetmutator.mutated_path = myapp:static/cache/
assetmutator.each_request = false
assetmutator.each_boot = true
assetmutator.asset_paths =
    myapp:static/js
    myapp:static/css
```

CHAPTER 7

Asset Concatenation (a.k.a Asset Pipeline)

A feature that is popular in some web frameworks (e.g. Ruby on Rails) is the ability to combine all assets that share a common type into a single file for sourcing within your views. Unfortunately, this functionality is currently beyond the scope of `pyramid_assetmutator`. Please have a look at the pyramid_fantastic and pyramid_webassets packages instead.

CHAPTER 8

More Information

8.1 pyramid_assetmutator API

assign_assetmutator(*config*, *ext*, *cmd*, *new_ext*)

Configuration method to set up/assign an asset mutator. This allows the various `assetmutator_*` view helper methods to know which mutator to run for a specified asset path.

Parameters

- **ext** (*string - Required*) – The file extension this mutator should match (e.g. coffee).
- **cmd** (*string - Required*) – The command to run (e.g. coffee -c -p). The filename to be mutated will automatically be appended to the end of this string when running the command.
- **new_ext** (*string - Required*) – The extension that the mutated filename should have (e.g. js).

> **Warning:** The specified mutator command must be installed, must be executable by the Pyramid process, and must *output the mutated data to stdout*. The last point can get tricky depending on the command, so be sure to check its command switches for the appropriate option.

For example, a mutator that would run `.coffee` files through the `coffee` command (compiling them into JavaScript) would look like:

```
config.assign_assetmutator('coffee', 'coffee -c -p', 'js')
```

assetmutator_url(*path*, ***kw*)

Returns a Pyramid `static_url()` of the mutated asset (and mutates the asset if needed).

Parameters

- **path** (*string - Required*) – The Pyramid asset path to process.
- **mutator** (*dict or string - Optional*) – Allows you to override/specify a specific mutator to use (e.g. `coffee`), or assign a brand new mutator dictionary to be used (e.g. `{'cmd': 'lessc', 'ext': 'css'}`)

assetmutator_path(*path*, ***kw*)

Returns a Pyramid `static_path()` of the mutated asset (and mutates the asset if needed).

Parameters

- **path** (*string - Required*) – The Pyramid asset path to process.

- **mutator** (*dict or string - Optional*) – Allows you to override/specify a specific mutator to use (e.g. `coffee`), or assign a brand new mutator dictionary to be used (e.g. `{'cmd': 'lessc', 'ext': 'css'}`)

assetmutator_source (*path*, ***kw*)

Returns the source data/contents of the mutated asset (and mutates the asset if needed). This is useful when you want to output inline data (e.g. for inline JavaScript blocks).

Parameters

- **path** (*string - Required*) – The Pyramid asset path to process.
- **mutator** (*dict or string - Optional*) – Allows you to override/specify a specific mutator to use (e.g. `coffee`), or assign a brand new mutator dictionary to be used (e.g. `{'cmd': 'lessc', 'ext': 'css'}`)

Note: Many template packages escape output by default. Consult your template language's syntax to output an unescaped string.

---

assetmutator_assetpath (*path*, ***kw*)

Returns a Pyramid asset specification such as `pkg:static/path/to/file.ext` (and mutates the asset if needed).

Parameters

- **path** (*string - Required*) – The Pyramid asset path to process.
- **mutator** (*dict or string - Optional*) – Allows you to override/specify a specific mutator to use (e.g. `coffee`), or assign a brand new mutator dictionary to be used (e.g. `{'cmd': 'lessc', 'ext': 'css'}`)

This function could be used to nest `pyramid_assetmutator` calls. e.g. `assetmutator_path(assetmutator_assetpath('pkg:static/js/script.coffee'))` could compile a CoffeeScript file into JS, and then further minify the JS file if your mutator configuration looked something like:

```
config.assign_assetmutator('coffee', 'coffee -c -p', 'js')
config.assign_assetmutator('js', 'uglifyjs', 'js')
```

includeme (*config*)

Activate the package; typically called via `config.include('pyramid_assetmutator')` instead of being invoked directly.

CHAPTER 9

Development Versions / Reporting Issues

Visit http://github.com/seedifferently/pyramid_assetmutator to download development or tagged versions.

Visit http://github.com/seedifferently/pyramid_assetmutator/issues to report issues.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

p

A

I

P