pyramid_assetmutator Documentation

Release 1.0b1

Seth Davis

February 22, 2017

Contents

CHAPTER 1

Overview

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

Its goal is to provide Pyramid developers with a basic and straightforward mechanism for utilizing asset *compilation* (e.g. for CoffeeScript/Sass), *minification* (e.g. with jsmin), and *optimization* (e.g. with pngcrush).

As of version 0.3, it also adds experimental support for template language parsing (e.g. you could use Pyramid helpers like `request.route_url()` within your CoffeeScript files by using `application.coffee.pt` as the asset source filename).

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 newly injected *`assign_assetmutator()`* configuration method, so that your application can know what kind of assets you'll be asking it to mutate. The configuration syntax for your Pyramid project's `__init__.py` is:

```
config.assign_assetmutator('SOURCE EXTENSION', 'COMMAND', 'OUTPUT EXTENSION')
```

For example, the following configuration would activate `pyramid_assetmutator` in your app, 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 included the module and configured your mutators, you will then be able to call one of the following view helper methods in your templates to *reference* (with Pyramid's asset specification syntax) and *"mutate"* (if needed) an asset:

class AssetMutator(*request*, *rendering_val*)

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')
```

Template Language Parsing

In version 0.3, experimental support for template language parsing was added. As long as the template language is known to Pyramid (e.g. one of these bindings has been configured), you can append the expected template filename extension to your asset filename and it will attempt to parse it before mutation.

For example, if the `pyramid_jinja2` package was configured, you could specify an asset path to an asset named `application.coffee.jinja2` and `pyramid_assetmutator` would run it through the Jinja2 renderer before mutation.

Warning: Current support is experimental, and there are a few caveats:

1. You must specify a `mutated_path` in your configuration so that the intermediate-step sources can be stored and parsed from that directory.
2. Template parsing is currently only supported when using the `each_request` configuration (which is the default configuration).
3. If Pyramid's "reload_templates" setting is false, templates will _NOT_ be reprocessed.
4. Hopefully obvious, but if the asset you are parsing uses a syntax that conflicts with the template language's syntax, things probably won't work out very well for you.

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>
```

As of version 0.3, your asset source could be parsed with Chameleon as well:

```
<script src="${assetmutator_url('pkg:static/js/test.coffee.pt')}"
        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 be 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 configurations 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

While the default settings will probably be fine for most people, custom settings can be configured via your Pyramid application's `.ini` file (in the app section representing your Pyramid app) using the `assetmutator` key:

`assetmutator.remutate_check`

Default stat

Options exists | stat | checksum

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

`assetmutator.each_request`

Default true

Whether or not assets should be checked/mutated during each request (whenever one of the `assetmutator_*` methods is encountered).

`assetmutator.each_boot`

Default []

Defines a list of asset specifications that should be checked/mutated when the application boots (uses Pyramid's `ApplicationCreated` event).

Limited "globbing" support is available (via the `glob` module), although checks are not recursive so you must be explicit.

e.g.:

```
assetmutator.each_boot =
    myapp:static/js/application.coffee
    myapp:static/css/*.sass
    myapp:static/css/admin/*.sass
```

`assetmutator.mutated_file_prefix`

Default _

A prefix to add to the mutated asset's output filename.

`assetmutator.mutated_path`

Default None

By default, mutated output files are stored in the same directory as their source files. If you would like to keep all mutated files 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.purge_mutated_path`

Default false

When `true`, if a valid `mutated_path` is specified then any files within it will be deleted when the application boots (uses Pyramid's `ApplicationCreated` event).

`assetmutator.always_remutate`

Default []

Defines a list of asset specifications that should *always* be remutated — even if the mutated version of the asset is already present.

Limited "globbing" support is available (via the `fnmatch` module), so a value of `*.sass` would match all Sass sources, while a star value (`*`) would specify that *all* sources should always be remutated.

e.g.:

```
assetmutator.always_remutate =
    *.sass
    myapp:static/js/application.coffee
```

Note: Combining this with the `each_request` setting can be useful in development environments when your source files contain imports and therefore may not always change but should still be remutated so that import changes are processed. However, this can significantly affect performance so it should only be utilized in environments that require it. Alternatively, you may use a `remutate_check` value of `stat` (the default) or `checksum` and manually "touch" a source file to trigger a remutate on the next request.

Production Example

As an example, if you wanted to only check/mutate assets on each boot (a good practice for production environments), processing CoffeeScript and Sass files in the `js` and `css` root and admin directories, with each mutated `_filename` stored in a `myapp:static/cache/` directory, your `.ini` file would look something like:

```
[app:main]
...other settings...
assetmutator.each_request = false
assetmutator.each_boot =
    myapp:static/js/*.coffee
    myapp:static/js/admin/*.coffee
    myapp:static/css/*.sass
    myapp:static/css/admin/*.sass
assetmutator.mutated_path = myapp:static/cache/
```

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 templates. However, this functionality is currently beyond the scope of `pyramid_assetmutator` as we consider it to have less and less relevance in an HTTP/2 era.

CHAPTER 8

More Information

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')
```

includeme(*config*)

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

class **AssetMutator**(*request*, *rendering_val*)

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')
```

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