Pipeline Documentation

Release 1.2.24

Timothée Peignier

January 13, 2013

CONTENTS

Pipeline is an asset packaging library for Django, providing both CSS and JavaScript concatenation and compression, built-in JavaScript template support, and optional data-URI image and font embedding.

You can report bugs and discuss features on the issues page.

You can discuss features or ask questions on the IRC channel on freenode : #django-pipeline

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

1. Either check out Pipeline from GitHub or to pull a release off PyPI

   ```
   pip install django-pipeline
   ```

2. Add 'pipeline' to your `INSTALLED_APPS`

   ```
   INSTALLED_APPS = (
       'pipeline',
   )
   ```

3. Use a pipeline storage for `STATICFILES_STORAGE`

   ```
   STATICFILES_STORAGE = 'pipeline.storage.PipelineCachedStorage'
   ```

Note: You need to use `Django>=1.4` or `django-staticfiles>=1.2.1` to be able to use this version of pipeline.

1.1.1 Recommendations

Pipeline's default CSS and JS compressor is Yuglify. Yuglify wraps UglifyJS and cssmin, applying the default YUI configurations to them. It can be downloaded from: https://github.com/yui/yuglify/.

If you do not install yuglify, make sure to disable the compressor in your settings.

1.2 Configuration

Configuration and list of available settings for Pipeline

1.2.1 Specifying files

You specify groups of files to be compressed in your settings. You can use glob syntax to select multiples files.

The basic syntax for specifying CSS/JavaScript groups files is

```
PIPELINE_CSS = {
    'colors': {
        'source_filenames': (
          'css/core.css',
          'css/colors/*.css',
          'css/layers.css'
        ),
        'output_filename': 'css/colors.css',
        'extra_context': {
            'media': 'screen,projection',
        },
    },
}

PIPELINE_JS = {
    'stats': {
        'source_filenames': (
          'js/jquery.js',
          'js/d3.js',
          'js/collections/*.js',
          'js/application.js',
        ),
        'output_filename': 'js/stats.js',
    }
}
```

Group options

source_filenames

Required

Is a tuple with the source files to be compressed. The files are concatenated in the order specified in the tuple.

output_filename

Required

Is the filename of the (to be) compressed file.

variant

Optional

Is the variant you want to apply to your CSS. This allow you to embed images and fonts in CSS with data-URI. Allowed values are : `None` and `datauri`.

Defaults to `None`.

template_name

Optional

Name of the template used to render `<script>` for js package or `<link>` for css package.

Defaults to `None`.

extra_context

Optional

Is a dictionary of values to add to the template context, when generating the HTML for the HTML-tags with the templatetags.

For CSS, if you do not specify `extra_context/media`, the default media in the `<link>` output will be `media="all"`.

manifest

Optional

Indicate if you want this group to appear in your cache manifest.

Defaults to `True`.

Other settings

PIPELINE

When `PIPELINE` is `True`, CSS and JavaScripts will be concatenated and filtered. When `False`, the source-files will be used instead.

Defaults to `not DEBUG` (compressed files will only be used when `DEBUG` is `False`).

PIPELINE_CSS_COMPRESSOR

Compressor class to be applied to CSS files.

If empty or `None`, CSS files won't be compressed.

Defaults to `'pipeline.compressors.yuglify.YuglifyCompressor'`.

PIPELINE_JS_COMPRESSOR

Compressor class to be applied to JavaScript files.

If empty or `None`, JavaScript files won't be compressed.

Defaults to `'pipeline.compressors.yuglify.YuglifyCompressor'`

Note: Please note that in order to use Yuglify compressor, you need to install Yuglify (see *Installation* for more details).

PIPELINE_TEMPLATE_NAMESPACE

Object name where all of your compiled templates will be added, from within your browser. To access them with your own JavaScript namespace, change it to the object of your choice.

Defaults to `"window.JST"`

PIPELINE_TEMPLATE_EXT

The extension for which Pipeline will consider the file as a Javascript template. To use a different extension, like `.mustache`, set this settings to `.mustache`.

Defaults to `".jst"`

PIPELINE_TEMPLATE_FUNC

JavaScript function that compiles your JavaScript templates. Pipeline doesn't bundle a javascript template library, but the default setting is to use the underscore template function.

Defaults to `"_.template"`

1.2.2 Embedding fonts and images

You can embed fonts and images directly in your compiled css, using Data-URI in modern browsers.

To do so, setup variant group options to the method you wish to use :

```
PIPELINE_CSS = {
    'master': {
        'source_filenames': (
          'css/core.css',
          'css/button/*.css',
        ),
        'output_filename': 'css/master.css',
        'variant': 'datauri',
    },
}
```

Images and fonts are embedded following these rules :

- If asset is under **32 kilobytes** to avoid rendering delay or not rendering at all in Internet Explorer 8.
- If asset path contains a directory named "**embed**".

Overriding embedding settings

You can override these rules using the following settings:

PIPELINE_EMBED_MAX_IMAGE_SIZE

Setting that controls the maximum image size (in bytes) to embed in CSS using Data-URIs. Internet Explorer 8 has issues with assets under 32 kilobytes.

Defaults to `32700`

PIPELINE_EMBED_PATH

Setting the directory that an asset needs to be in so that it is embedded

Defaults to `r'[/]?embed/'`

1.2.3 Rewriting CSS urls

If the source CSS contains relative URLs (i.e. relative to current file), those URLs will be converted to full relative path.

1.2.4 Wrapped javascript output

All javascript output is wrapped in an anonymous function :

```
(function(){ ... })();
```

This safety wrapper, make it difficult to pollute the global namespace by accident and improve performance.

You can override this behavior by setting `PIPELINE_DISABLE_WRAPPER` to `True`.

1.3 Usage

Describes how to use Pipeline when it is installed and configured.

1.3.1 Templatetags

Pipeline includes two template tags: `compressed_css` and `compressed_js`, in a template library called `compressed`.

They are used to output the `<link>` and `<script>`-tags for the specified CSS/JavaScript-groups (as specified in the settings). The first argument must be the name of the CSS/JavaScript group.

The templatetags will either output the source filenames or the compressed filenames, depending on the `PIPELINE` setting, if you do not specify the `PIPELINE` setting, the source files will be used in DEBUG-mode, and compressed files in non-DEBUG-mode.

If you need to change the output of the HTML-tags generated from the templatetags, this can be done by overriding the templates `pipeline/css.html` and `pipeline/js.html`.

Example

If you have specified the CSS-groups "screen" and "print" and a JavaScript-group with the name "scripts", you would use the following code to output them all

```
{% load compressed %}
{% compressed_css 'colors' %}
{% compressed_js 'stats' %}
```

1.3.2 Collect static

Pipeline integrates with staticfiles, you just need to setup `STATICFILES_STORAGE` to

```
STATICFILES_STORAGE = 'pipeline.storage.PipelineStorage'
```

Then when you run `collectstatic` command, your CSS and your javascripts will be compressed in the same time

```
$ python oslo/manage.py collectstatic
```

1.3.3 Middleware

To enable HTML compression add `pipeline.middleware.MinifyHTMLMiddleware`, to your `MIDDLEWARE_CLASSES` settings.

Ensure that it comes after any middleware which modifies your HTML, like `GZipMiddleware`

```
MIDDLEWARE_CLASSES = (
   'django.middleware.gzip.GZipMiddleware',
   'pipeline.middleware.MinifyHTMLMiddleware',
)
```

1.3.4 Cache manifest

Pipeline provide a way to add your javascripts and stylesheets files to a cache-manifest via Manifesto.

To do so, you just need to add manifesto app to your `INSTALLED_APPS`.

1.3.5 Jinja2

Pipeline also includes Jinja2 support and is used almost identically to the Django Template tags implementation.

Note: You have to expose the Jinja2 functions provided by pipeline to the Jinja2 environment yourself, Pipeline will not do this for you. There are several implementations of Jinja2 for Django, like `django-ninja` or `coffin`.

See the vendor documentation for examples on how to expose functions to the Jinja2 environment and pick a solution that best suites your use case.

For more information on Jinja2 see the documentation at http://jinja.pocoo.org/docs/.

Functions

The functions to expose to the Jinja2 environment are:

```
pipeline.jinja2.ext.compressed_css
pipeline.jinja2.ext.compressed_js
```

Example

To use in the templates:

```
{{ compressed_css('group_name') }}
{{ compressed_js('group_name') }}
```

Templates

Unlike the Django template tag implementation the Jinja2 implementation uses different templates, so if you wish to override them please override `pipeline/css.jinja` and `pipeline/js.jinja`.

1.4 Compressors

1.4.1 Yuglify compressor

The Yuglify compressor uses yuglify for compressing javascript and stylesheets.

To use it for your stylesheets add this to your `PIPELINE_CSS_COMPRESSOR`

```
PIPELINE_CSS_COMPRESSOR = 'pipeline.compressors.yuglify.YuglifyCompressor'
```

To use it for your javascripts add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = 'pipeline.compressors.yuglify.YuglifyCompressor'
```

PIPELINE_YUGLIFY_BINARY

Command line to execute for the Yuglify program. You will most likely change this to the location of yuglify on your system.

Defaults to `'/usr/bin/env yuglify'`.

PIPELINE_YUGLIFY_CSS_ARGUMENTS

Additional arguments to use when compressing CSS.

Defaults to `''`.

PIPELINE_YUGLIFY_JS_ARGUMENTS

Additional arguments to use when compressing JavaScript.

Defaults to `''`.

1.4.2 YUI Compressor compressor

The YUI compressor uses yui-compressor for compressing javascript and stylesheets.

To use it for your stylesheets add this to your `PIPELINE_CSS_COMPRESSOR`

```
PIPELINE_CSS_COMPRESSOR = 'pipeline.compressors.yui.YUICompressor'
```

To use it for your javascripts add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = 'pipeline.compressors.yui.YUICompressor'
```

PIPELINE_YUI_BINARY

Command line to execute for the YUI program. You will most likely change this to the location of yuicompressor on your system.

Defaults to `'/usr/bin/env yuicompressor'`.

Warning: Don't point to `yuicompressor.jar` directly, we expect to find a executable script.

PIPELINE_YUI_CSS_ARGUMENTS

Additional arguments to use when compressing CSS.

Defaults to `''`.

PIPELINE_YUI_JS_ARGUMENTS

Additional arguments to use when compressing JavaScript.

Defaults to `''`.

1.4.3 Closure Compiler compressor

The Closure compressor uses Google Closure Compiler to compress javascripts.

To use it add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = (
  'pipeline.compressors.closure.ClosureCompressor',
)
```

PIPELINE_CLOSURE_BINARY

Command line to execute for the Closure Compiler program. You will most likely change this to the location of closure on your system.

Default to `'/usr/bin/env closure'`

Warning: Don't point to `compiler.jar` directly, we expect to find a executable script.

PIPELINE_CLOSURE_ARGUMENTS

Additional arguments to use when closure is called.

Default to `''`

1.4.4 UglifyJS compressor

The UglifyJS compressor uses UglifyJS to compress javascripts.

To use it add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = 'pipeline.compressors.uglifyjs.UglifyJSCompressor'
```

PIPELINE_UGLIFYJS_BINARY

Command line to execute for the Closure Compiler program. You will most likely change this to the location of closure on your system.

Defaults to `'/usr/bin/env uglifyjs'`.

PIPELINE_UGLIFYJS_ARGUMENTS

Additional arguments to use when uglifyjs is called.

Default to `''`

1.4.5 JSMin compressor

The jsmin compressor uses Douglas Crockford jsmin tool to compress javascripts.

To use it add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = 'pipeline.compressors.jsmin.JSMinCompressor'
```

Install the jsmin library with your favorite Python package manager

```
pip install jsmin
```

1.4.6 SlimIt compressor

The slimit compressor uses SlimIt to compress javascripts.

To use it add this to your `PIPELINE_JS_COMPRESSOR`

```
PIPELINE_JS_COMPRESSOR = 'pipeline.compressors.slimit.SlimItCompressor'
```

Install the slimit library with your favorite Python package manager

```
pip install slimit
```

1.4.7 CSSTidy compressor

The CSStidy compressor uses CSStidy to compress stylesheets.

To us it for your stylesheets add this to your `PIPELINE_CSS_COMPRESSOR`

```
PIPELINE_CSS_COMPRESSOR = 'pipeline.compressors.csstidy.CSSTidyCompressor'
```

PIPELINE_CSSTIDY_BINARY

Command line to execute for csstidy program. You will most likely change this to the location of csstidy on your system.

Defaults to `'/usr/bin/env csstidy'`

PIPELINE_CSSTIDY_ARGUMENTS

Additional arguments to use when csstidy is called.

Default to `'--template=highest'`

1.4.8 CSSMin compressor

The cssmin compressor uses the cssmin command to compress stylesheets. To use it, add this to your PIPELINE_CSS_COMPRESSOR

```
PIPELINE_CSS_COMPRESSOR = 'pipeline.compressors.cssmin.CSSMinCompressor'
```

PIPELINE_CSSMIN_BINARY

Command line to execute for cssmin program. You will most likely change this to the location of cssmin on your system.

Defaults to `'/usr/bin/env cssmin'`

PIPELINE_CSSMIN_ARGUMENTS

Additional arguments to use when cssmin is called.

Default to `''`

1.4.9 Write your own compressor class

You can write your own compressor class, for example if you want to implement other types of compressors.

To do so, you just have to create a class that inherits from `pipeline.compressors.CompressorBase` and implements `compress_css` and/or a `compress_js` when needed.

Finally, add it to `PIPELINE_CSS_COMPRESSOR` or `PIPELINE_JS_COMPRESSOR` settings (see *Configuration* for more information).

Example

A custom compressor for an imaginary compressor called jam

```
from pipeline.compressors import CompressorBase

class JamCompressor(CompressorBase):
  def compress_js(self, js):
    return jam.compress(js)
```

```
  def compress_css(self, css):
    return jam.compress(css)
```

Add it to your settings

```
PIPELINE_CSS_COMPRESSOR = 'jam.compressors.JamCompressor'
PIPELINE_JS_COMPRESSOR = 'jam.compressors.JamCompressor'
```

1.5 Compilers

1.5.1 Coffee Script compiler

The Coffee Script compiler uses Coffee Script to compile your javascripts.

To use it add this to your PIPELINE_COMPILERS

```
PIPELINE_COMPILERS = (
  'pipeline.compilers.coffee.CoffeeScriptCompiler',
)
```

PIPELINE_COFFEE_SCRIPT_BINARY

Command line to execute for coffee program. You will most likely change this to the location of coffee on your system.

Defaults to '/usr/bin/env coffee'.

PIPELINE_COFFEE_SCRIPT_ARGUMENTS

Additional arguments to use when coffee is called.

Defaults to ''.

1.5.2 LESS compiler

The LESS compiler uses LESS to compile your stylesheets.

To use it add this to your PIPELINE_COMPILERS

```
PIPELINE_COMPILERS = (
  'pipeline.compilers.less.LessCompiler',
)
```

PIPELINE_LESS_BINARY

Command line to execute for lessc program. You will most likely change this to the location of lessc on your system.

Defaults to '/usr/bin/env lessc'.

PIPELINE_LESS_ARGUMENTS

Additional arguments to use when lessc is called.

Defaults to ''.

1.5.3 SASS compiler

The SASS compiler uses SASS to compile your stylesheets.

To use it add this to your PIPELINE_COMPILERS

```
PIPELINE_COMPILERS = (
  'pipeline.compilers.sass.SASSCompiler',
)
```

PIPELINE_SASS_BINARY

Command line to execute for sass program. You will most likely change this to the location of sass on your system.

Defaults to '/usr/bin/env sass'.

PIPELINE_SASS_ARGUMENTS

Additional arguments to use when sass is called.

Defaults to ''.

1.5.4 Stylus compiler

The Stylus compiler uses *Stylus <http://learnboost.github.com/stylus/>* to compile your stylesheets.

To use it add this to your PIPELINE_COMPILERS

```
PIPELINE_COMPILERS = (
    'pipeline.compilers.stylus.StylusCompiler',
)
```

PIPELINE_STYLUS_BINARY

Command line to execute for stylus program. You will most likely change this to the location of stylus on your system.

Defaults to '/usr/bin/env stylus'.

PIPELINE_STYLUS_ARGUMENTS

Additional arguments to use when stylus is called.

Defaults to ''.

1.5.5 Write your own compiler class

You can write your own compiler class, for example if you want to implement other types of compilers.

To do so, you just have to create a class that inherits from `pipeline.compilers.CompilerBase` and implements `match_file` and `compile_file` when needed.

Finally, specify it in the tuple of compilers `PIPELINE_COMPILERS` in the settings.

Example

A custom compiler for an imaginary compiler called jam

```
from pipeline.compilers import CompilerBase

class JamCompiler(CompilerBase):
  output_extension = 'js'

  def match_file(self, filename):
    return filename.endswith('.jam')

  def compile_file(self, infile, outfile, outdated=False, force=False):
    if not outdated and not force:
      return  # No need to recompiled file
    return jam.compile(infile, outfile)
```

1.6 Javascript Templates

Pipeline allows you to use javascript templates along with your javascript views. To use your javascript templates, just add them to your `PIPELINE_JS` group

```
PIPELINE_JS = {
  'application': {
    'source_filenames': (
      'js/application.js',
      'js/templates/**/*.jst',
    ),
    'output_filename': 'js/application.js'
  }
}
```

For example, if you have the following template `js/templates/photo/detail.jst`

```
<div class="photo">
 <img src="<%= src %>" />
 <div class="caption">
  <%= caption %>
 </div>
</div>
```

It will be available from your javascript code via window.JST

```
JST.photo_detail({ src:"images/baby-panda.jpg", caption:"A baby panda is born" });
```

1.6.1 Configuration

Template function

By default, Pipeline uses a variant of Micro Templating to compile the templates, but you can choose your preferred JavaScript templating engine by changing `PIPELINE_TEMPLATE_FUNC`

```
PIPELINE_TEMPLATE_FUNC = 'template'
```

Template namespace

Your templates are made available in a top-level object, by default `window.JST`, but you can choose your own via `PIPELINE_TEMPLATE_NAMESPACE`

```
PIPELINE_TEMPLATE_NAMESPACE = 'window.Template'
```

Template extension

Templates are detected by their extension, by default `.jst`, but you can use your own extension via `PIPELINE_TEMPLATE_EXT`

```
PIPELINE_TEMPLATE_EXT = '.mustache'
```

1.6.2 Using it with your favorite template library

Mustache

To use it with Mustache you will need this some extra javascript

```
Mustache.template = function(templateString) {
  return function() {
    if (arguments.length < 1) {
      return templateString;
    } else {
      return Mustache.to_html(templateString, arguments[0], arguments[1]);
    }
  };
};
```

And use this settings

```
PIPELINE_TEMPLATE_EXT = '.mustache'
PIPELINE_TEMPLATE_FUNC = 'Mustache.template'
```

Prototype

To use it with Prototype, just setup your `PIPELINE_TEMPLATE_FUNC`

```
PIPELINE_TEMPLATE_FUNC = 'new Template'
```

1.7 Storages

1.7.1 Using with a custom storage

Pipeline uses Django Storage to read, save and delete files, by default it use an improved `StaticFilesStorage`.

You can provide your own via `PIPELINE_STORAGE` :

```
PIPELINE_STORAGE = 'storages.backends.s3boto.S3BotoStorage'
```

1.7.2 Using with staticfiles

Pipeline is providing a storage for staticfiles app, to use it configure `STATICFILES_STORAGE` like so

```
STATICFILES_STORAGE = 'pipeline.storage.PipelineStorage'
```

And if you want versioning use

```
STATICFILES_STORAGE = 'pipeline.storage.PipelineCachedStorage'
```

There is also non-packing storage available, that allows you to run `collectstatic` command without packaging your assets. Useful for production when you don't want to run compressor or compilers

```
STATICFILES_STORAGE = 'pipeline.storage.NonPackagingPipelineStorage'
```

Also available if you want versioning

```
STATICFILES_STORAGE = 'pipeline.storage.NonPackagingPipelineCachedStorage'
```

Pipeline is also providing a storage that play nicely with staticfiles app particularly for development :

```
PIPELINE_STORAGE = 'pipeline.storage.PipelineFinderStorage'
```

1.7.3 Using with other storages

You can also use your own custom storage, for example, if you want to use S3 for your assets :

```
STATICFILES_STORAGE = 'your.app.S3PipelineStorage'
```

Your storage only need to inherit from `PipelineMixin` and/or `CachedFilesMixin` :

```
from staticfiles.storage import CachedFilesMixin

from pipeline.storage import PipelineMixin

from storages.backends.s3boto import S3BotoStorage


class S3PipelineStorage(PipelineMixin, CachedFilesMixin, S3BotoStorage):
    pass
```

1.8 Signals

List of all signals sent by pipeline.

1.8.1 css_compressed

pipeline.signals.css_compressed

Whenever a css package is compressed, this signal is sent after the compression.

Arguments sent with this signal :

sender The `Packager` class that compressed the group.

package The package actually compressed.

1.8.2 js_compressed

pipeline.signals.js_compressed

Whenever a js package is compressed, this signal is sent after the compression.

Arguments sent with this signal :

sender The `Packager` class that compressed the group.

package The package actually compressed.

1.9 Sites using Pipeline

The following sites are a partial list of people using Pipeline.

Are you using pipeline and not being in this list? Drop us a line.

1.9.1 20 Minutes

For their internal tools: http://www.20minutes.fr

1.9.2 The Molly Project

Molly is a framework for the rapid development of information and service portals targeted at mobile internet devices: http://mollyproject.org

It powers the University of Oxford's mobile portal: http://m.ox.ac.uk/

1.9.3 Croisé dans le Métro

For their main and mobile website:

- http://www.croisedanslemetro.com
- http://m.croisedanslemetro.com

1.9.4 Ulule

For their main and forum website:

- http://www.ulule.com
- http://vox.ulule.com

CHAPTER

TWO

INDICES AND TABLES

- *search*