Gears Documentation

Release 0.5.1

Mike Yumatov

October 16, 2012

CONTENTS

Gears is a library to compile, concatenate and minify JavaScript and CSS assets, highly inspired by Ruby's Sprockets. You can also write scripts, styles and client templates using CoffeeScript, Handlebars, Stylus, Less, and compile them using external packages (gears-coffeescript, gears-handlebars, gears-stylus, gears-less). These packages already include all required node.js modules, so you don't need to worry about installing them yourself.

There is also:

- django-gears, an app for Django that integrates Gears with Django project;
- Flask-Gears, an extension that integrates Gears with Flask application;
- gears-cli, a command-line utility that compiles assets. It also can watch assets for changes and automaticaly re-compile them.

CHAPTER

ONE

SOURCE CODE

Gears code is hosted on GitHub: https://github.com/gears/gears.

CHAPTER

TWO

CONTENTS

2.1 About

2.1.1 The Problem

The amount of code on the client side significantly grows in the modern web-applications. JavaScript and CSS files size increases, and it becomes more difficult to navigate through them. Static files must be broken down into modules. But the more static files you connect to the HTML page, the more HTTP requests must be done to load this page, which increases the load time.

CSS and JavaScript files must be combined in production to reduce the number of subsequent HTTP requests to load the page. This is what Gears does for you.

2.1.2 Motivation and Design Decisions

But this problem is not new, and there are many awesome Python libraries here to solve it. So why another one? I've tried almost every existent library, and none of them fits my needs (and taste). Some of them are only for Django, some require you to specify asset dependencies in Python (or YAML, or JSON, or HTML).

So, when I decided to create Gears, I pursued two goals:

- this library should be framework-agnostic and cover as much as possible contexts;
- asset dependencies should be described in the usual way, much like this is done in the modern languages (e.g., in Python).

Let's look at both.

Usage Contexts

I have to work with static in different contexts:

- in Django projects;
- in reusable Django apps;
- in Flask apps;
- in static sites.

And I want to use only one library for all this contexts (I don't want to deal with many different libraries).

Asset Dependencies

Yes, there are already Python libraries, that cover all this contexts. But I don't like their approach to describing dependencies between assets. It should be more like how we import modules in Python, Ruby, Java, etc. Dependencies for the asset should be described in this asset, not in the other place. Imagine for a moment that all imports in Python project would have to be described in JSON file in the root of this project. It would be terrible.

I like how this problem is solved in Ruby's Sprockets library. Dependencies between assets must be described in header comments with special syntax. It was decided to use this approach and syntax in Gears.

2.2 Installation

You can install Gears with pip:

```
$ pip install Gears
```

If you want to use node.js-dependent compilers or compressors, you need to install other dependencies:

```
$ pip install gears-less          # LESS
$ pip install gears-stylus        # Stylus
$ pip install gears-handlebars    # Handlebars
$ pip install gears-coffeescript  # CoffeeScript

$ pip install gears-uglifyjs      # UglifyJS
$ pip install gears-clean-css     # clean-css
```

Please note that all these compilers and compressors require node.js to be installed on your system.

It is strongly recommended to install Gears within activated virtualenv.

If you want to use one of available extensions (django-gears, Flask-Gears or gears-cli), please refer to its documentation instead.

2.2.1 Installing the Development Version

If you want to work with the latest version of Gears, install it from the public repository (Git is required):

```
$ pip install -e git+https://github.com/gears/gears@develop#egg=Gears
```

2.3 API

2.3.1 Environment

class gears.environment.**Environment**(*root*, *public_assets=(<function <lambda> at 0x3fc0a28>, '^css/style\.css$', '^js/script\.js$')*, *cache=None*)

This is the central object, that links all Gears parts. It is passed the absolute path to the directory where public assets will be saved. Environment contains registries for file finders, compilers, compressors, processors and supported MIME types.

Parameters

- **root** – the absolute path to the directory where handled public assets will be saved by `save()` method.

- **public_assets** – a list of public assets paths.
- **cache** – a cache object. It is used by assets and dependencies to store compilation results.

compilers = None
The registry for asset compilers. See `Compilers` for more information.

compressors = None
The registry for asset compressors. See `Compressors` for more information.

find(*item*, *logical=False*)
Find files using `finders` registry. The `item` parameter can be an instance of `AssetAttributes` class, a path to the asset or a logical path to the asset. If `item` is a logical path, *logical* parameter must be set to `True`.

Returns a tuple with `AssetAttributes` instance for found file path as first item, and absolute path to this file as second item.

If nothing is found, `gears.exceptions.FileNotFound` exception is rased.

finders = None
The registry for file finders. See `Finders` for more information.

list(*path*, *mimetype=None*, *recursive=False*)
Yield two-tuples for all files found in the directory given by `path` parameter. Result can be filtered by the second parameter, `mimetype`, that must be a MIME type of assets compiled source code. If `recursive` is `True`, then `path` will be scanned recursively. Each tuple has `AssetAttributes` instance for found file path as first item, and absolute path to this file as second item.

Usage example:

```
# Yield all files from 'js/templates' directory.
environment.list('js/libs')

# Yield only files that are in 'js/templates' directory and have
# 'application/javascript' MIME type of compiled source code.
environment.list('js/templates', mimetype='application/javascript')
```

mimetypes = None
The registry for supported MIME types. See `MIMETypes` for more information.

postprocessors = None
The registry for asset postprocessors. See `Postprocessors` for more information.

preprocessors = None
The registry for asset preprocessors. See `Preprocessors` for more information.

register_defaults()
Register default compilers, preprocessors and MIME types.

save()
Save handled public assets to `root` directory.

suffixes
The registry for supported suffixes of assets. It is built from MIME types and compilers registries, and is cached at the first call. See `Suffixes` for more information.

File Finders Registry

class `gears.environment.`**Finders**
The registry for file finders. This is just a list of finder objects. Each finder object must be an instance of any

`BaseFinder` subclass. Finders from this registry are used by `Environment` object in the order they were added.

register(*finder*)
Append passed `finder` to the list of finders.

unregister(*finder*)
Remove passed `finder` from the list of finders. If `finder` does not found in the registry, nothing happens.

MIME Types Registry

class `gears.environment.`**MIMETypes**
The registry for MIME types. It acts like a dict with extensions as keys and MIME types as values. Every registered extension can have only one MIME type.

register(*extension*, *mimetype*)
Register passed `mimetype` MIME type with `extension` extension.

register_defaults()
Register MIME types for `.js` and `.css` extensions.

unregister(*extension*)
Remove registered MIME type for passed `extension` extension. If MIME type for this extension does not found in the registry, nothing happens.

Compilers Registry

class `gears.environment.`**Compilers**
The registry for compilers. It acts like a dict with extensions as keys and compilers as values. Every registered extension can have only one compiler.

register(*extension*, *compiler*)
Register passed *compiler* with passed *extension*.

unregister(*extension*)
Remove registered compiler for passed *extension*. If compiler for this extension does not found in the registry, nothing happens.

Preprocessors Registry

class `gears.environment.`**Preprocessors**
The registry for asset preprocessors. It acts like a dictionary with MIME types as keys and lists of processors as values. Every registered MIME type can have many preprocessors. Preprocessors for the MIME type are used in the order they were added.

register_defaults()
Register `DirectivesProcessor` as a preprocessor for *text/css* and *application/javascript* MIME types.

Postprocessors Registry

class `gears.environment.`**Postprocessors**
The registry for asset postprocessors. It acts like a dictionary with MIME types as keys and lists of processors

as values. Every registered MIME type can have many postprocessors. Postprocessors for the MIME type are used in the order they were added.

Compressors Registry

class `gears.environment.`**`Compressors`**
The registry for asset compressors. It acts like a dictionary with MIME types as keys and compressors as values. Every registered MIME type can have only one compressor.

`register`(*mimetype*, *compressor*)
Register passed *compressor* for passed *mimetype*.

`unregister`(*mimetype*)
Remove registered compressors for passed *mimetype*. If compressor for this MIME type does not found in the registry, nothing happens.

Suffixes Registry

class `gears.environment.`**`Suffixes`**
The registry for asset suffixes. It acts like a list of dictionaries. Every dictionary has three keys: `extensions`, `result_mimetype` and `mimetype`:

- `suffix` is a suffix as a list of extensions (e.g. `['.js', '.coffee']`);
- `result_mimetype` is a MIME type of a compiled asset with this suffix;
- `mimetype` is a MIME type, for which this suffix is registered.

2.3.2 Asset Attributes

class `gears.asset_attributes.`**`AssetAttributes`**(*environment*, *path*)
Provides access to asset path properties. The attributes object is created with environment object and relative (or logical) asset path.

Some properties may be useful or not, depending on the type of passed path. If it is a relative asset path, you can use all properties except `search_paths`. In case of a logical asset path it makes sense to use only those properties that are not related to processors and compressor.

Parameters

- **environment** – an instance of `Environment` class.
- **path** – a relative or logical path of the asset.

`compiler_extensions`
The list of compiler extensions. Example:

```
>>> attrs = AssetAttributes(environment, 'js/lib/external.min.js.coffee')
>>> attrs.compiler_extensions
['.coffee']
```

`compiler_format_extension`
Implicit format extension on the asset by its compilers.

`compiler_mimetype`
Implicit MIME type of the asset by its compilers.

`compilers`
The list of compilers used to build asset.

compressor
The compressors used to compress the asset.

dirname = None
The relative path to the directory the asset.

environment = None
Used to access the registries of compilers, processors, etc. It can be also used by asset. See `Environment` for more information.

extensions
The list of asset extensions. Example:

```
>>> attrs = AssetAttributes(environment, 'js/models.js.coffee')
>>> attrs.extensions
['.js', '.coffee']

>>> attrs = AssetAttributes(environment, 'js/lib/external.min.js.coffee')
>>> attrs.format_extension
['.min', '.js', '.coffee']
```

format_extension
The format extension of asset. Example:

```
>>> attrs = AssetAttributes(environment, 'js/models.js.coffee')
>>> attrs.format_extension
'.js'

>>> attrs = AssetAttributes(environment, 'js/lib/external.min.js.coffee')
>>> attrs.format_extension
'.js'
```

logical_path
The logical path to asset. Example:

```
>>> attrs = AssetAttributes(environment, 'js/models.js.coffee')
>>> attrs.logical_path
'js/models.js'
```

mimetype
MIME type of the asset.

path = None
The relative (or logical) path to asset.

path_without_suffix
The relative path to asset without suffix. Example:

```
>>> attrs = AssetAttributes(environment, 'js/app.js')
>>> attrs.path_without_suffix
'js/app'
```

postprocessors
The list of postprocessors used to build asset.

preprocessors
The list of preprocessors used to build asset.

processors
The list of all processors (preprocessors, compilers, postprocessors) used to build asset.

search_paths
The list of logical paths which are used to search for an asset. This property makes sense only if the attributes was created with logical path.

It is assumed that the logical path can be a directory containing a file named `index` with the same suffix.

Example:

```
>>> attrs = AssetAttributes(environment, 'js/app.js')
>>> attrs.search_paths
['js/app.js', 'js/app/index.js']

>>> attrs = AssetAttributes(environment, 'js/app/index.js')
>>> attrs.search_paths
['js/models/index.js']
```

suffix
The list of asset extensions starting from the format extension. Example:

```
>>> attrs = AssetAttributes(environment, 'js/lib/external.min.js.coffee')
>>> attrs.suffix
['.js', '.coffee']
```

2.3.3 Asset Handlers

class gears.asset_handler.**BaseAssetHandler**
Base class for all asset handlers (processors, compilers and compressors). A subclass has to implement `__call__()` which is called with asset as argument.

__call__(*asset*)
Subclasses have to override this method to implement the actual handler function code. This method is called with asset as argument. Depending on the type of the handler, this method must change asset state (as it does in `Directivesprocessor`) or return some value (in case of asset compressors).

classmethod as_handler(***initkwargs*)
Converts the class into an actual handler function that can be used when registering different types of processors in `Environment` class instance.

The arguments passed to `as_handler()` are forwarded to the constructor of the class.

class gears.asset_handler.**ExecMixin**
Provides the ability to process asset through external command.

executable = None
The name of the executable to run. It must be a command name, if it is available in the PATH environment variable, or a path to the executable.

get_args()
Returns the list of `subprocess.Popen` arguments.

get_process()
Returns `subprocess.Popen` instance with args from `get_args()` result and piped stdin, stdout and stderr.

params = []
The list of executable parameters.

run(*input*)
Runs `executable` with `input` as stdin. `AssetHandlerError` exception is raised, if execution is failed, otherwise stdout is returned.

Processors

class gears.processors.**BaseProcessor**
Base class for all asset processors. Subclass's `__call__()` method must change asset's `processed_source` attribute.

Compilers

class gears.compilers.**BaseCompiler**
Base class for all asset compilers. Subclass's `__call__()` method must change asset's `processed_source` attribute.

result_mimetype = None
MIME type of the asset source code after compiling.

2.4 Changelog

2.4.1 0.5.1 (2012-10-16)

- Fix saving handled assets.
- Python 3.3 is also supported.

2.4.2 0.5 (2012-10-16)

- Support for Python 3.2 was added (Thanks to Artem Gluvchynsky).

Note: SlimIt and cssmin compressors don't support Python 3 yet. But you can use compressors from gears-uglifyjs and gears-clean-css packages instead.

2.4.3 0.4 (2012-09-23)

- Public assets storage was simplified. There is no registry for them anymore. They are set using `public_assets` param of `Environment` now.

  Also, public assets handling was slightly improved. `public_assets` must be a list or tuple of callables or regexps now. Default value:

```
DEFAULT_PUBLIC_ASSETS = (
    lambda path: not any(path.endswith(ext) for ext in ('.css', '.js')),
    r'^css/style\.css$',
    r'^js/script\.js$',
)
```

  `css/style.css`, `js/script.js` and all assets that aren't compiled to .css or .js are public by default.
- Added `require_tree` directive. It works like `require_directory`, but also collects assets from subdirectories recursively.
- Node.js-dependent compilers (CoffeeScript, Handlebars, Stylus and LESS) and compressors (UglifyJS and clean-css) have been moved into separate packages (gears-coffeescript, gears-handlebars, gears-stylus, gears-less, gears-uglifyjs, gears-clean-css), as they required some additional work to make them work (install some

node.js modules, point your app to them, etc.). Now all these packages already include all required node.js modules, so you don't need to worry about installing them yourself.

- SASS and SCSS compilers have been removed since they did nothing to really support SASS and SCSS compilation.
- Support for Python 2.5 was dropped.

2.4.4 0.3 (2012-06-24)

- Added `depend_on` directive. It is useful when you need to specify files that affect an asset, but not to include them into bundled asset or to include them using compilers. E.g., if you use `@import` functionality in some CSS pre-processors (Less or Stylus).
- Main extensions (`.js` or `.css`) can be omitted now in asset file names. E.g., you can rename `application.js.coffee` asset to `application.coffee`.
- Asset requirements are restricted by MIME type now, not by extension. E.g., you can require Handlebars templates or JavaScript assets from CoffeeScript now.
- Added file-based cache.
- Environment cache is pluggable now.
- Fixed cache usage in assets.

2.4.5 0.2 (2012-02-18)

- Fix `require_directory` directive, so it handles removed/renamed/added assets correctly. Now it adds required directory to asset's dependencies set.
- Added asset dependencies. They are not included to asset's bundled source, but if dependency is expired, then asset is expired. Any file of directory can be a dependency.
- Cache is now asset agnostic, so other parts of Gears are able to use it.
- Added support for SlimIt as JavaScript compressor.
- Added support for cssmin as CSS compressor.
- Refactored compressors, compilers and processors. They are all subclasses of `BaseAssetHandler` now.
- Added config for Travis CI.
- Added some docs.
- Added more tests.

2.4.6 0.1.1 (2012-02-26)

- Added missing files to MANIFEST.in

2.4.7 0.1 (2012-02-26)

First public release.

PYTHON MODULE INDEX

g