Improve your user experience with amazingly fast page loads by combining, compressing, and versioning your JavaScript & CSS files and images. django-mediagenerator eliminates unnecessary HTTP requests and maximizes cache usage. Also, it provides lots of advanced features required for building HTML5 web applications (e.g. HTML5 offline manifests, Sass support, etc.).

Take a look at the feature comparison for a quick overview and if you like django-mediagenerator please click the I use this! button on that page. Thank you!

Django mediagenerator lives on GitHub (downloads, source code and bug tracking).
1.1 settings

1.1.1 MEDIA_BUNDLES

This defines all JavaScript and CSS bundles as a list of tuples where the first tuple entry denotes the bundle output name and the following tuple entries denote the input file names:

```
MEDIA_BUNDLES = (
    ('main.css', # bundle name
        # input files
        'css/reset.css',
        'css/design.css',
    ),
    ('main.js',
        'js/jquery.js',
        'js/jquery.autocomplete.js',
    ),
    # ...
)
```

Internally, all input file names are converted to filters. Instead of file names you can also be more explicit and specify filters as dicts:

```
MEDIA_BUNDLES = (
    ('main.css',
        {'filter': 'mediagenerator.generators.bundles.base.FileFilter',
         'name': 'css/reset.css'},
    ),
    ('main.js',
        {'filter': 'mediagenerator.generators.bundles.base.FileFilter',
         'name': 'css/design.css'},
    ),
    # ...
)
```

The dict notation allows for using advanced features and configuring individual filters. You will rarely need to use it explicitly, though.

1.1.2 ROOT_MEDIA_FILTERS

Defines which filters should be applied bundles of a certain file type. Note that these filters are applied after the input files got combined/bundled and thus is primarily useful for compressors and URL rewriters and other post-processing filters. Filters on input files need to be specified as documented in MEDIA_BUNDLES.
This is a dict where the key denotes the file type and the value is either a string or a tuple of strings denoting the filters that should be applied:

```python
ROOT_MEDIA_FILTERS = {
    'js': 'mediagenerator.filters.closure.Closure',
}
```

Filter names are always converted to dicts. The filter above in combination with the MEDIA_BUNDLES setting from above would become:

```python
MEDIA_BUNDLES = {
    'main.js': {
        'filter': 'mediagenerator.filters.closure.Closure',
        'input': ['js/jquery.js', 'js/jQuery.autocomplete.js'],
    }
}
```

Internally, the media generator uses a few additional filters which get inserted between the input files and ROOT_MEDIA_FILTERS. For example, one of those filters takes care of concatenating/combining all input files.

1.1.3 MEDIA_GENERATORS

All low-level backends derive from the `mediagenerator.base.Generator` class. This setting defines the list of generators used in your project. Its default value is:

```python
MEDIA_GENERATORS = ('mediagenerator.generators.copyfiles.CopyFiles',
                    'mediagenerator.generators.bundles.Bundles',
                    'mediagenerator.generators.manifest.Manifest',
)
```

The Bundles generator in that list takes care of handling MEDIA_BUNDLES. As you can see, even bundle handling is just a normal generator backend. Note that bundle filters which are used in MEDIA_BUNDLES are a different type of backend. The Bundle generator provides a `mediagenerator.generators.bundles.base.Filter` backend class for those.

1.1.4 GLOBAL_MEDIA_DIRS

Tuple of paths which should be added to the media search path.

1.1.5 IGNORE_APP_MEDIA_DIRS

Tuple of app names for which the “static” folder shouldn’t be added to the media search path.

1.1.6 DEV_MEDIA_URL

The URL for serving media when MEDIA_DEV_MODE is False.
1.1.7 PRODUCTION_MEDIA_URL

The URL for serving media when MEDIA_DEV_MODE is True.

1.1.8 MEDIA_DEV_MODE

A boolean which defines whether we’re on the development or production server. If True media files aren’t combined and compressed in order to simplify debugging.

1.1.9 Settings for non-standard project structures

These settings might be required for certain “franchised” project structures. The need for these settings is an indicator that maybe (!) you’re not using a clean project structure and that the dependencies in your project might be turned upside down. Nevertheless, there are blog posts endorsing such a structure and people with a pretty large code base depend on it. So, here are the settings for overriding the output locations of the generated media files:

- GENERATED_MEDIA_DIR: Overrides the path of the _generated_media folder.
- GENERATED_MEDIA_NAMES_MODULE: Overrides the import path of the _generated_media_names module.
- GENERATED_MEDIA_NAMES_FILE: Overrides the path (on the file system) to the _generated_media_names module.

1.2 Templatetags

The media template library contains all tags needed for working with media files. You can load the library like this:

```
{% load media %}
```

1.2.1 Including JS and CSS

You can include JS and CSS bundles using:

```html
<head>
  ...
  {% include_media 'main.css' %}
  ...
</head>
```

The include_media tag automatically generates the required <link> or <script> HTML code for the respective bundle. In production it generates just a single tag. In development mode it generates multiple tags, one for each file that is part of the bundle.

You can optionally specify the CSS media type via:

```html
{% include_media 'main.css' media='screen,print' %}
```

1.2.2 Including images and other files

Image URLs can be generated using:
The `media_url` tag only works with assets that consist of a single file (e.g., an image or an HTML offline manifest). It does not work with bundles or other assets which generate into multiple URLs either in development or production mode.

### 1.3 Filter and Generators

#### 1.3.1 Image/file copying

**Generator:** mediagenerator.generators.copyfiles.CopyFiles

You can define the file extensions that should be copied into the `_generated_media` folder via the `COPY_MEDIA_FILETYPES` setting which is a tuple of file extensions. Example:

```python
COPY_MEDIA_FILETYPES = ('gif', 'jpg', 'jpeg', 'png', 'svg', 'svgz', 'ico', 'swf', 'ttf', 'otf', 'eot')
```

By default, images, Flash files, and fonts are copied. Additionally, you can specify a tuple of file name regexes that should be ignored via `IGNORE_MEDIA_COPY_PATTERNS`. All copied files will have a version hash in their file name.

#### 1.3.2 Closure Compiler

**Filter:** mediagenerator.filters.closure.Closure

Compresses your JavaScript files via Google’s Closure Compiler.

Installation in `settings.py`:

```python
ROOT_MEDIA_FILTERS = {
    'js': 'mediagenerator.filters.closure.Closure',
}
CLOSURE_COMPILER_PATH = '/path/to/closure/compiler'
```

You can also define the compilation level via `CLOSURE_COMPILATION_LEVEL`. By default this is set to `'SIMPLE_OPTIMIZATIONS'`.

#### 1.3.3 YUICompressor

**Filter:** mediagenerator.filters.yuicompressor.YUICompressor

Compresses your JavaScript and CSS files via YUICompressor.

Installation in `settings.py`:

```python
ROOT_MEDIA_FILTERS = {
    'css': 'mediagenerator.filters.yuicompressor.YUICompressor',
    'js': 'mediagenerator.filters.yuicompressor.YUICompressor',
}
YUICOMPRESSOR_PATH = '/path/to/yuicompressor'
```
1.3.4 Sass/Compass

**Filter:** mediagenerator.filters.sass.Sass

Sass files are automatically detected by their file extension. Simply mention .sass files in MEDIA_BUNDLES exactly like you would with .css files.

It’s possible to use features from Compass and its extensions. Run manage.py importsassframeworks to add the respective files to your project. Extensions can be listed via SASS_FRAMEWORKS in settings.py. For example, this is how you’d add ninesixty and susy in addition to the default frameworks (compass and blueprint):

```python
SASS_FRAMEWORKS = {
    'compass',
    'blueprint',
    'ninesixty',
    'susy',
}
```

Note that you have to install the Compass binary even if you don’t use any Sass/Compass frameworks in your project.

If you use the FireSass Firebug plugin you should set SASS_DEBUG_INFO = True in your settings, so additional debug information is emitted for FireSass.

See also: Using Sass with django-mediagenerator

1.3.5 Less CSS

**Filter:** mediagenerator.filters.lesscss.LessCSS

Less files are automatically detected by their file extension. Simply mention .less files in MEDIA_BUNDLES exactly like you would with .css files.

Less CSS plugin also comes with LESS watch support.

Include the Less.js file somewhere within your static dirs, and create a bundle with it, and enable LESS CSS debugging.

Note that LESS CSS debugging only works in dev mode.

In settings.py:

```python
MEDIA_BUNDLES = {
    #define any bundle with LESS CSS files
    ('main.css',
    'less/reset.less',
    'less/fonts.less',
    'less/common.less',
    'less/structure.less',
    'less/tabbox.less',
    'less/sliders.less',
    ),
    #include Less in a bundle
    ('less.js',
    'js/less.js',
    ),
}

ENABLE_LESS_CSS_DEBUG = True
```

In your base.html (or wherever you include your bundles), use this code to initialize less.js

1.3. Filter and Generators
@import syntax has limited support in this release.

### 1.3.6 CleverCSS

**Filter:** mediagenerator.filters.clevercss.CleverCSS

CleverCSS files are automatically detected by their file extension. Simply mention `.ccss` files in `MEDIA_BUNDLES` exactly like you would with `.css` files.

### 1.3.7 CoffeeScript

**Filter:** mediagenerator.filters.coffeescript.CoffeeScript

CoffeeScript files are automatically detected by their file extension. Simply mention `.coffee` files in `MEDIA_BUNDLES` exactly like you would with `.js` files.

Literate CoffeeScript is also supported. Use the `.litcoffee` extension for those files in `MEDIA_BUNDLES`.

### 1.3.8 Accessing media URLs from JavaScript

**Filter:** mediagenerator.filters.media_url.MediaURL

Provides JavaScript functions for retrieving the URL of a media file, similar to the `{% media_url %}` template tag.

Installation in `settings.py`:

```python
MEDIA_BUNDLES = {
    'main.js',
    {'filter': 'mediagenerator.filters.media_url.MediaURL'},
    'js/jquery.js',
    'js/jquery.autocomplete.js',
    # ...
}
```
In your JavaScript code you’ll then have a `media_url()` function which returns the URL for a given file. Only files that exist in the `_generated_media` folder can be resolved this way.

If you try to resolve a bundle and the bundle consists of multiple files and `MEDIA_DEV_MODE` is `True` the `media_url()` function will return a list of (uncombined) URLs instead of a single string. Make sure that your code checks for this case.

### 1.3.9 PyvaScript

**Filter:** `mediagenerator.filters.pyvascript_filter.PyvaScript`

PyvaScript files are automatically detected by their file extension. Simply mention `.pyva` files in `MEDIA_BUNDLES` exactly like you would with `.js` files. The PyvaScript standard library can be integrated by using the file name `.stdlib.pyva`. Here is an example that integrates jQuery, PyvaScript’s standard library, and your own code (e.g. `yourcode.pyva`):

```python
MEDIA_BUNDLES = (  
    ('main.js',  
    'jquery.js',  
    '.stdlib.pyva',  
    'yourcode.pyva',  
    ),  
)
```

### 1.3.10 Python in the browser (TODO)

**Filter:** `mediagenerator.filters.pyjs_filter.Pyjs`

TODO: document me :)

See also:
- Offline HTML5 canvas app in Python with django-mediagenerator, Part 1: pyjs
- Offline HTML5 canvas app in Python with django-mediagenerator, Part 2: Drawing

### 1.3.11 HTML5 offline manifests (TODO)

**Generator:** `mediagenerator.generators.manifest.Manifest`

TODO: document me :)

See also: HTML5 offline manifests with django-mediagenerator

### 1.3.12 Data URIs / image sprites (TODO)

**Filter:** `mediagenerator.filters.cssurl.CSSURL`

**Filter:** `mediagenerator.filters.cssurl.CSSURLFileFilter`

**Generator:** `mediagenerator.generators.mhtml.MHTML (not yet implemented)`

TODO: document me and write a nice tutorial about me :)

---

1.3. Filter and Generators
1.3.13 Translations (i18n) bundling (TODOC)

Filter: mediagenerator.filters.i18n.I18N

TODO: document me and write a nice tutorial about me :)

1.3.14 Django templates (TODOC)

Filter: mediagenerator.filters.tempate.Template

Auto-applied to .html files. Uses Django’s template language to render the contents of the given file. Can also be specified manually for individual files with the explicit dict filter syntax (see MEDIA_BUNDLES). In that case, the input files are listed via input. TODO: document me :)

1.3.15 Combining files in dev mode

Filter: mediagenerator.filters.concat.Concat

Sometimes you might want to use the repository version of a certain JavaScript framework. Often, these frameworks consist of several individual files which have to be combined in order to work correctly. Since the media generator doesn’t combine files in development mode you might want to enforce concatenation.

This example shows how to combine some files of the XUI framework:

```python
MEDIA_BUNDLES = {
    'main.js',
    # XUI and its dependencies
    'emile.js',
    {'filter': 'mediagenerator.filters.concat.Concat',
     'dev_output_name': 'xui.js',
     'concat_dev_output': True,
     'input': {
         'xui/header.js',
         'xui/base.js',
         'xui/core/dom.js',
         'xui/core/event.js',
         'xui/core/fx.js',
         'xui/core/style.js',
         'xui/core/xhr.js',
         'xui/footer.js',
     }
}
```

The concat_dev_output option allows to enforce concatenation. The dev_output_name option allows you to specify a nice human-readable file name which will appear in URLs on the development server. This is recommended for debugging purposes.
• Getting started: django-mediagenerator: total asset management
• Using Sass with django-mediagenerator
• Offline HTML5 canvas app in Python with django-mediagenerator, Part 1: pyjs
• Offline HTML5 canvas app in Python with django-mediagenerator, Part 2: Drawing
• HTML5 offline manifests with django-mediagenerator
3.1 Q: How does it relate to django-staticfiles / django.contrib.staticfiles?

django-medialogenerator is a complete standalone asset manager which replaces django-staticfiles. You can still use both in the same project if you really need to, but that’s very rarely the case (e.g. during a gradual transition from django-staticfiles to django-medialogenerator).

3.2 Q: What are the perfect caching headers?

- Disable ETags because they cause unnecessary If-modified-since requests.
- Use Cache-Control: public, max-age=31536000

3.3 Tip: How to include IE-specific stylesheets

Imagine you have several stylesheets combined into main.css bundle. Now imagine you also have an extra stylesheet for Internet Explorer called ie.css. Most websites include their IE-specific stylesheet with an additional <link /> tag using conditional HTML. The problem with this solution is that IE users have to wait for two requests: one for main.css and another one for ie.css. Can this be done more efficiently?

Yes! Create two CSS bundles (main-ie.css with ie.css and main.css without ie.css) in your settings. For example:

```python
_base_main_bundle = {
    'css/reset.css',
    'css/design.css',
}

MEDIA_BUNDLES = {
    ('main.css',),
    + _base_main_bundle,
    ('main-ie.css',),
    + _base_main_bundle
    + ('ie.css',),
}
```

Then, use this conditional comment sequence to include the bundles:
Now every browser will only make one single request. Cool, isn’t it?

In the example above only IE6 and IE7 get special treatment. IE8 loads the same stylesheet as all other browsers. Of course you can extend the example to serve different stylesheets for all the different IE versions.