Pelican Documentation

Release 3

Alexis Métaireau

November 20, 2012

CONTENTS

Pelican is a static site generator, written in Python.

- Write your weblog entries directly with your editor of choice (vim!) in reStructuredText or Markdown
- Includes a simple CLI tool to (re)generate the weblog
- Easy to interface with DVCSes and web hooks
- Completely static output is easy to host anywhere

CHAPTER

ONE

FEATURES

Pelican currently supports:

- Blog articles and pages
- Comments, via an external service (Disqus). (Please note that while useful, Disqus is an external service, and thus the comment data will be somewhat outside of your control and potentially subject to data loss.)
- Theming support (themes are created using Jinja2 templates)
- PDF generation of the articles/pages (optional)
- Publication of articles in multiple languages
- Atom/RSS feeds
- Code syntax highlighting
- Compilation of LESS CSS (optional)
- Import from WordPress, Dotclear, or RSS feeds
- Integration with external tools: Twitter, Google Analytics, etc. (optional)

CHAPTER

TWO

WHY THE NAME “PELICAN”?

“Pelican” is an anagram for *calepin*, which means “notebook” in French. ;)

CHAPTER

THREE

SOURCE CODE

You can access the source code at: https://github.com/getpelican/pelican

CHAPTER

FOUR

FEEDBACK / CONTACT US

If you want to see new features in Pelican, don't hesitate to offer suggestions, clone the repository, etc. There are many ways to *contribute*. That's open source, dude!

Send a message to "alexis at notmyidea dot org" with any requests/feedback! You can also join the team at #pelican on Freenode (or if you don't have an IRC client handy, use the webchat for quick feedback.

CHAPTER

FIVE

DOCUMENTATION

A French version of the documentation is available at `fr/index`.

5.1 Getting started

5.1.1 Installing Pelican

You're ready? Let's go! You can install Pelican via several different methods. The simplest is via pip:

```
$ pip install pelican
```

If you don't have `pip` installed, an alternative method is `easy_install`:

```
$ easy_install pelican
```

While the above is the simplest method, the recommended approach is to create a virtual environment for Pelican via virtualenv and virtualenvwrapper before installing Pelican. Assuming you've followed the virtualenvwrapper installation and shell configuration steps, you can then open a new terminal session and create a new virtual environment for Pelican:

```
$ mkvirtualenv pelican
```

Once the virtual environment has been created and activated, Pelican can be be installed via `pip` or `easy_install` as noted above. Alternatively, if you have the project source, you can install Pelican using the distutils method:

```
$ cd path-to-Pelican-source
$ python setup.py install
```

If you have Git installed and prefer to install the latest bleeding-edge version of Pelican rather than a stable release, use the following command:

```
$ pip install -e git://github.com/getpelican/pelican#egg=pelican
```

If you plan on using Markdown as a markup format, you'll need to install the Markdown library as well:

```
$ pip install Markdown
```

Upgrading

If you installed a stable Pelican release via `pip` or `easy_install` and wish to upgrade to the latest stable release, you can do so by adding `--upgrade` to the relevant command. For pip, that would be:

```
$ pip install --upgrade pelican
```

If you installed Pelican via distutils or the bleeding-edge method, simply perform the same step to install the most recent version.

Dependencies

At this time, Pelican is dependent on the following Python packages:

- feedgenerator, to generate the Atom feeds
- jinja2, for templating support
- docutils, for supporting reStructuredText as an input format

If you're not using Python 2.7, you will also need the `argparse` package.

Optionally:

- pygments, for syntax highlighting
- Markdown, for supporting Markdown as an input format

5.1.2 Kickstart a blog

Following is a brief tutorial for those who want to get started right away. We're going to assume that virtualenv and virtualenvwrapper are installed and configured; if you've installed Pelican outside of a virtual environment, you can skip to the `pelican-quickstart` command. Let's first create a new virtual environment and install Pelican into it:

```
$ mkvirtualenv pelican
$ pip install pelican Markdown
```

Next we'll create a directory to house our site content and configuration files, which can be located any place you prefer, and associate this new project with the currently-active virtual environment:

```
$ mkdir ~/code/yoursitename
$ cd ~/code/yoursitename
$ setvirtualenvproject
```

Now we can run the `pelican-quickstart` command, which will ask some questions about your site:

```
$ pelican-quickstart
```

Once you finish answering all the questions, you can begin adding content to the *content* folder that has been created for you. (See *Writing articles using Pelican* section below for more information about how to format your content.) Once you have some content to generate, you can convert it to HTML via the following command:

```
$ make html
```

If you'd prefer to have Pelican automatically regenerate your site every time a change is detected (handy when testing locally), use the following command instead:

```
$ make regenerate
```

To serve the site so it can be previewed in your browser at http://localhost:8000:

```
$ make serve
```

Normally you would need to run `make regenerate` and `make serve` in two separate terminal sessions, but you can run both at once via:

```
$ make devserver
```

The above command will simultaneously run Pelican in regeneration mode as well as serve the output at http://localhost:8000. Once you are done testing your changes, you should stop the development server via:

```
$ ./develop_server.sh stop
```

When you're ready to publish your site, you can upload it via the method(s) you chose during the `pelican-quickstart` questionnaire. For this example, we'll use rsync over ssh:

```
$ make rsync_upload
```

That's it! Your site should now be live.

5.1.3 Writing articles using Pelican

File metadata

Pelican tries to be smart enough to get the information it needs from the file system (for instance, about the category of your articles), but some information you need to provide in the form of metadata inside your files.

You can provide this metadata in reStructuredText text files via the following syntax (give your file the `.rst` extension):

```
My super title
##############

:date: 2010-10-03 10:20
:tags: thats, awesome
:category: yeah
:author: Alexis Metaireau
```

Pelican implements an extension of reStructuredText to enable support for the `abbr` HTML tag. To use it, write something like this in your post:

```
This will be turned into :abbr:`HTML (HyperText Markup Language)`.
```

You can also use Markdown syntax (with a file ending in `.md`). Markdown generation will not work until you explicitly install the `Markdown` package, which can be done via `pip install Markdown`. Metadata syntax for Markdown posts should follow this pattern:

```
Date: 2010-12-03
Title: My super title
Tags: thats, awesome
Slug: my-super-post

This is the content of my super blog post.
```

Note that, aside from the title, none of this metadata is mandatory: if the date is not specified, Pelican will rely on the file's "mtime" timestamp, and the category can be determined by the directory in which the file resides. For example, a file located at `python/foobar/myfoobar.rst` will have a category of `foobar`.

Generate your blog

The `make` shortcut commands mentioned in the `Kickstart a blog` section are mostly wrappers around the `pelican` command that generates the HTML from the content. The `pelican` command can also be run directly:

```
$ pelican /path/to/your/content/ [-s path/to/your/settings.py]
```

The above command will generate your weblog and save it in the `content/` folder, using the default theme to produce a simple site. The default theme is simple HTML without styling and is provided so folks may use it as a basis for creating their own themes.

Pelican has other command-line switches available. Have a look at the help to see all the options you can use:

```
$ pelican --help
```

Auto-reload

It's possible to tell Pelican to watch for your modifications, instead of manually re-running it every time you want to see your changes. To enable this, run the `pelican` command with the `-r` or `--autoreload` option.

Pages

If you create a folder named `pages`, all the files in it will be used to generate static pages.

Then, use the `DISPLAY_PAGES_ON_MENU` setting, which will add all the pages to the menu.

If you want to exclude any pages from being linked to or listed in the menu then add a `status: hidden` attribute to its metadata. This is useful for things like making error pages that fit the generated theme of your site.

Importing an existing blog

It is possible to import your blog from Dotclear, WordPress, and RSS feeds using a simple script. See *Import from other blog software*.

Translations

It is possible to translate articles. To do so, you need to add a `lang` meta attribute to your articles/pages and set a `DEFAULT_LANG` setting (which is English [en] by default). With those settings in place, only articles with the default language will be listed, and each article will be accompanied by a list of available translations for that article.

Pelican uses the article's URL "slug" to determine if two or more articles are translations of one another. The slug can be set manually in the file's metadata; if not set explicitly, Pelican will auto-generate the slug from the title of the article.

Here is an example of two articles, one in English and the other in French.

The English article:

```
Foobar is not dead
##################

:slug: foobar-is-not-dead
:lang: en

That's true, foobar is still alive!
```

And the French version:

```
Foobar n'est pas mort !
#######################

:slug: foobar-is-not-dead
:lang: fr

Oui oui, foobar est toujours vivant !
```

Post content quality notwithstanding, you can see that only item in common between the two articles is the slug, which is functioning here as an identifier. If you'd rather not explicitly define the slug this way, you must then instead ensure that the translated article titles are identical, since the slug will be auto-generated from the article title.

Syntax highlighting

Pelican is able to provide colorized syntax highlighting for your code blocks. To do so, you have to use the following conventions (you need to put this in your content files).

For RestructuredText:

```
.. code-block:: identifier

   your code goes here
```

For Markdown, format your code blocks thusly:

```
:::identifier
your code goes here
```

The specified identifier should be one that appears on the list of available lexers.

Publishing drafts

If you want to publish an article as a draft (for friends to review before publishing, for example), you can add a `status: draft` attribute to its metadata. That article will then be output to the `drafts` folder and not listed on the index page nor on any category page.

Viewing the generated files

The files generated by Pelican are static files, so you don't actually need anything special to see what's happening with the generated files.

You can either use your browser to open the files on your disk:

```
$ firefox output/index.html
```

Or run a simple web server using Python:

```
cd output && python -m SimpleHTTPServer
```

5.2 Settings

Pelican is configurable thanks to a configuration file you can pass to the command line:

```
$ pelican -s path/to/your/settingsfile.py path
```

Settings are configured in the form of a Python module (a file). You can see an example by looking at /samples/pelican.conf.py

All the setting identifiers must be set in all-caps, otherwise they will not be processed. Setting values that are numbers (5, 20, etc.), booleans (True, False, None, etc.), dictionaries, or tuples should *not* be enclosed in quotation marks. All other values (i.e., strings) *must* be enclosed in quotation marks.

The settings you define in the configuration file will be passed to the templates, which allows you to use your settings to add site-wide content.

Here is a list of settings for Pelican:

5.2.1 Basic settings

Setting name (default value)	What does it do?
AUTHOR	Default author (put your name)
DATE_FORMATS (`{}`)	If you do manage multiple languages, you can set the date formatting here. See “Date format and locales” section below for details.
DEFAULT_CATEGORY (`'misc'`)	The default category to fall back on.
DEFAULT_DATE_FORMAT (`'%a %d %B %Y'`)	The default date format you want to use.
DISPLAY_PAGES_ON_MENU (`True`)	Whether to display pages on the menu of the template. Templates may or not honor this setting.
DEFAULT_DATE (`fs`)	The default date you want to use. If ‘fs’, Pelican will use the file system timestamp information (mtime) if it can’t get date information from the metadata. If tuple object, it will instead generate the default datetime object by passing the tuple to the datetime.datetime constructor.
JINJA_EXTENSIONS (`[]`)	A list of any Jinja2 extensions you want to use.
DELETE_OUTPUT_DIRECTORY (`False`)	Delete the content of the output directory before generating new files.
LOCALE (“[1])	Change the locale. A list of locales can be provided here or a single string representing one locale. When providing a list, all the locales will be tried until one works.
MARKUP (`('rst', 'md')`)	A list of available markup languages you want to use. For the moment, the only available values are *rst* and *md*.
MD_EXTENSIONS (`['codehilite','extra']`)	A list of the extensions that the Markdown processor will use. Refer to the extensions chapter in the Python-Markdown documentation for a complete list of supported extensions.
OUTPUT_PATH (`'output/'`)	Where to output the generated files.
PATH (`None`)	Path to look at for input files.
PAGE_DIR (`'pages'`)	Directory to look at for pages.
PAGE_EXCLUDES (`()`)	A list of directories to exclude when looking for pages.
ARTICLE_DIR (”)	Directory to look at for articles.
ARTICLE_EXCLUDES: (`('pages',)`)	A list of directories to exclude when looking for articles.
PDF_GENERATOR (`False`)	Set to True if you want to have PDF versions of your documents. You will need to install *rst2pdf*.
RELATIVE_URLS (`True`)	Defines whether Pelican should use document-relative URLs or not. If set to `False`, Pelican will use the SITEURL setting to construct absolute URLs.
PLUGINS (`[]`)	The list of plugins to load. See *Plugins*.
SITENAME (`'A Pelican Blog'`)	Your site name
SITEURL	Base URL of your website. Not defined by default, so it is best to specify your SITEURL; if you do not, feeds will not be generated with properly-formed URLs. You should include `http://` and your domain, with no trailing slash at the end. Example: `SITEURL = 'http://mydomain.com'`
STATIC_PATHS (`['images']`)	The static paths you want to have accessible on the output path “static”. By default, Pelican will copy the ‘images’ folder to the output folder.
TIMEZONE	The timezone used in the date information, to generate Atom and RSS feeds. See the “timezone” section below for more info.
TYPOGRIFY (`False`)	If set to True, several typographical improvements will be incorporated into the generated HTML via the Typogrify library, which can be installed via: `pip install typogrify`
LESS_GENERATOR (`FALSE`)	Set to True or complete path to *lessc* (if not found in system PATH) to enable compiling less css files. Requires installation of less css.
DIRECT_TEMPLATES (`('index', 'tags', 'categories', 'archives')`)	List of templates that are used directly to render content. Typically direct templates are used to generate index pages for collections of content e.g. tags and category index pages.
PAGINATED_DIRECT_TEMPLATES (`('index',)`)	Provides the direct templates that should be paginated.

URL settings

The first thing to understand is that there are currently two supported methods for URL formation: *relative* and *absolute*. Document-relative URLs are useful when testing locally, and absolute URLs are reliable and most useful when publishing. One method of supporting both is to have one Pelican configuration file for local development and another for publishing. To see an example of this type of setup, use the `pelican-quickstart` script as described at the top of the *Getting Started* page, which will produce two separate configuration files for local development and publishing, respectively.

You can customize the URLs and locations where files will be saved. The URLs and SAVE_AS variables use Python's format strings. These variables allow you to place your articles in a location such as '{slug}/index.html' and link to them as '{slug}' for clean URLs. These settings give you the flexibility to place your articles and pages anywhere you want.

Note: If you specify a datetime directive, it will be substituted using the input files' date metadata attribute. If the date is not specified for a particular file, Pelican will rely on the file's mtime timestamp.

Check the Python datetime documentation at http://bit.ly/cNcJUC for more information.

Also, you can use other file metadata attributes as well:

- slug
- date
- lang
- author
- category

Example usage:

- ARTICLE_URL = 'posts/{date:%Y}/{date:%b}/{date:%d}/{slug}/'
- ARTICLE_SAVE_AS = 'posts/{date:%Y}/{date:%b}/{date:%d}/{slug}/index.html'

This would save your articles in something like '/posts/2011/Aug/07/sample-post/index.html', and the URL to this would be '/posts/2011/Aug/07/sample-post/'.

Setting name (default value)	what does it do?
ARTICLE_URL ('{slug}.html')	The URL to refer to an ARTICLE.
ARTICLE_SAVE_AS ('{slug}.html')	The place where we will save an article.
ARTICLE_LANG_URL ('{slug}-{lang}.html')	The URL to refer to an ARTICLE which doesn't use the default language.
ARTICLE_LANG_SAVE_AS ('{slug}-{lang}.html'	The place where we will save an article which doesn't use the default language.
PAGE_URL ('pages/{slug}.html')	The URL we will use to link to a page.
PAGE_SAVE_AS ('pages/{slug}.html')	The location we will save the page.
PAGE_LANG_URL ('pages/{slug}-{lang}.html')	The URL we will use to link to a page which doesn't use the default language.
PAGE_LANG_SAVE_AS ('pages/{slug}-{lang}.html')	The location we will save the page which doesn't use the default language.
AUTHOR_URL ('author/{name}.html')	The URL to use for an author.
AUTHOR_SAVE_AS ('author/{name}.html')	The location to save an author.
CATEGORY_URL ('category/{name}.html')	The URL to use for a category.
CATEGORY_SAVE_AS ('category/{name}.html')	The location to save a category.
TAG_URL ('tag/{name}.html')	The URL to use for a tag.
TAG_SAVE_AS ('tag/{name}.html')	The location to save the tag page.
<DIRECT_TEMPLATE_NAME>_SAVE_AS	The location to save content generated from direct templates. Where <DIRECT_TEMPLATE_NAME> is the upper case template name.

Note: When any of **_SAVE_AS* is set to False, files will not be created.

Timezone

If no timezone is defined, UTC is assumed. This means that the generated Atom and RSS feeds will contain incorrect date information if your locale is not UTC.

Pelican issues a warning in case this setting is not defined, as it was not mandatory in previous versions.

Have a look at the wikipedia page to get a list of valid timezone values.

Date format and locale

If no DATE_FORMAT is set, fall back to DEFAULT_DATE_FORMAT. If you need to maintain multiple languages with different date formats, you can set this dict using language name (`lang` in your posts) as key. Regarding available format codes, see strftime document of python :

```
DATE_FORMAT = {
    'en': '%a, %d %b %Y',
    'jp': '%Y-%m-%d(%a)',
}
```

You can set locale to further control date format:

```
LOCALE = ('usa', 'jpn',   # On Windows
    'en_US', 'ja_JP'      # On Unix/Linux
    )
```

Also, it is possible to set different locale settings for each language. If you put (locale, format) tuples in the dict, this will override the LOCALE setting above:

```
# On Unix/Linux
DATE_FORMAT = {
    'en': ('en_US','%a, %d %b %Y'),
    'jp': ('ja_JP','%Y-%m-%d(%a)'),
}

# On Windows
DATE_FORMAT = {
    'en': ('usa','%a, %d %b %Y'),
    'jp': ('jpn','%Y-%m-%d(%a)'),
}
```

This is a list of available locales on Windows . On Unix/Linux, usually you can get a list of available locales via the `locale -a` command; see manpage locale(1) for more information.

5.2.2 Feed settings

By default, Pelican uses Atom feeds. However, it is also possible to use RSS feeds if you prefer.

Pelican generates category feeds as well as feeds for all your articles. It does not generate feeds for tags by default, but it is possible to do so using the `TAG_FEED_ATOM` and `TAG_FEED_RSS` settings:

Setting name (default value)	What does it do?
FEED_DOMAIN (`None`, i.e. base URL is "/")	The domain prepended to feed URLs. Since feed URLs should always be absolute, it is highly recommended to define this (e.g., "http://feeds.example.com"). If you have already explicitly defined SITEURL (see above) and want to use the same domain for your feeds, you can just set: *FEED_DOMAIN = SITEURL*
FEED_ATOM (`'feeds/all.atom.xml'`)	Relative URL to output the Atom feed.
FEED_RSS (`None`, i.e. no RSS)	Relative URL to output the RSS feed.
CATEGORY_FEED_ATOM ('feeds/%s.atom.xml'[2])	Where to put the category Atom feeds.
CATEGORY_FEED_RSS (`None`, i.e. no RSS)	Where to put the category RSS feeds.
TAG_FEED_ATOM (`None`, i.e. no tag feed)	Relative URL to output the tag Atom feed. It should be defined using a "%s" match in the tag name.
TAG_FEED_RSS (`None`, ie no RSS tag feed)	Relative URL to output the tag RSS feed
FEED_MAX_ITEMS	Maximum number of items allowed in a feed. Feed item quantity is unrestricted by default.

If you don't want to generate some of these feeds, set `None` to the variables above. If you don't want to generate any feeds set both `FEED_ATOM` and `FEED_RSS` to none.

[2] %s is the name of the category.

FeedBurner

If you want to use FeedBurner for your feed, you will likely need to decide upon a unique identifier. For example, if your site were called "Thyme" and hosted on the www.example.com domain, you might use "thymefeeds" as your unique identifier, which we'll use throughout this section for illustrative purposes. In your Pelican settings, set the *FEED_ATOM* attribute to "thymefeeds/main.xml" to create an Atom feed with an original address of *http://www.example.com/thymefeeds/main.xml*. Set the *FEED_DOMAIN* attribute to *http://feeds.feedburner.com*, or *http://feeds.example.com* if you are using a CNAME on your own domain (i.e., FeedBurner's "MyBrand" feature).

There are two fields to configure in the FeedBurner interface: "Original Feed" and "Feed Address". In this example, the "Original Feed" would be *http://www.example.com/thymefeeds/main.xml* and the "Feed Address" suffix would be *thymefeeds/main.xml*.

5.2.3 Pagination

The default behaviour of Pelican is to list all the article titles along with a short description on the index page. While it works pretty well for small-to-medium blogs, for sites with large quantity of articles it would be convenient to have a way to paginate the list.

You can use the following settings to configure the pagination.

Setting name (default value)	What does it do?
DEFAULT_ORPHANS (0)	The minimum number of articles allowed on the last page. Use this when you don't want to have a last page with very few articles.
DEFAULT_PAGINATION (False)	The maximum number of articles to include on a page, not including orphans. False to disable pagination.

5.2.4 Tag cloud

If you want to generate a tag cloud with all your tags, you can do so using the following settings.

Setting name (default value)	What does it do?
TAG_CLOUD_STEPS (4)	Count of different font sizes in the tag cloud.
TAG_CLOUD_MAX_ITEMS (100)	Maximum number of tags in the cloud.

The default theme does not support tag clouds, but it is pretty easy to add:

```
<ul>
    {% for tag in tag_cloud %}
        <li class="tag-{{ tag.1 }}"><a href="/tag/{{ tag.0 }}/">{{ tag.0 }}</a></li>
    {% endfor %}
</ul>
```

You should then also define a CSS style with the appropriate classes (tag-0 to tag-N, where N matches *TAG_CLOUD_STEPS* -1).

5.2.5 Translations

Pelican offers a way to translate articles. See the Getting Started section for more information.

Setting name (default value)	What does it do?
DEFAULT_LANG (`'en'`)	The default language to use.
TRANSLATION_FEED ('feeds/all-%s.atom.xml'[3])	Where to put the feed for translations.

5.2.6 Ordering content

Setting name (default value)	What does it do?
NEWEST_FIRST_ARCHIVES (`True`)	Order archives by newest first by date. (False: orders by date with older articles first.)
REVERSE_CATEGORY_ORDER (`False`)	Reverse the category order. (True: lists by reverse alphabetical order; default lists alphabetically.)

5.2.7 Theming

Theming is addressed in a dedicated section (see *How to create themes for Pelican*). However, here are the settings that are related to theming.

Setting name (default value)	What does it do?
THEME	Theme to use to produce the output. Can be the complete static path to a theme folder, or chosen between the list of default themes (see below)
THEME_STATIC_PATHS (`['static']`)	Static theme paths you want to copy. Default value is *static*, but if your theme has other static paths, you can put them here.
CSS_FILE (`'main.css'`)	Specify the CSS file you want to load.
WEBASSETS (`False`)	Asset management with *webassets* (see below)

By default, two themes are available. You can specify them using the *-t* option:

- notmyidea
- simple (a synonym for "full text" :)

You can define your own theme too, and specify its placement in the same manner. (Be sure to specify the full absolute path to it.)

Here is *a guide on how to create your theme*

You can find a list of themes at http://github.com/getpelican/pelican-themes.

Pelican comes with *pelican-themes*, a small script for managing themes.

The *notmyidea* theme can make good use of the following settings. I recommend using them in your themes as well.

[3] %s is the language

Setting name	What does it do ?
DISQUS_SITENAME	Pelican can handle Disqus comments. Specify the Disqus sitename identifier here.
GITHUB_URL	Your GitHub URL (if you have one). It will then use this information to create a GitHub ribbon.
GOOGLE_ANALYTICS	'UA-XXXX-YYYY' to activate Google Analytics.
GOSQUARED_SITENAME	'XXX-YYYYYY-X' to activate GoSquared.
MENUITEMS	A list of tuples (Title, URL) for additional menu items to appear at the beginning of the main menu.
PIWIK_URL	URL to your Piwik server - without 'http://' at the beginning.
PIWIK_SSL_URL	If the SSL-URL differs from the normal Piwik-URL you have to include this setting too. (optional)
PIWIK_SITE_ID	ID for the monitored website. You can find the ID in the Piwik admin interface > settings > websites.
LINKS	A list of tuples (Title, URL) for links to appear on the header.
SOCIAL	A list of tuples (Title, URL) to appear in the "social" section.
TWITTER_USERNAME	Allows for adding a button to articles to encourage others to tweet about them. Add your Twitter username if you want this button to appear.

In addition, you can use the "wide" version of the *notmyidea* theme by adding the following to your configuration:

```
CSS_FILE = "wide.css"
```

Asset management

The *WEBASSETS* setting allows to use the webassets module to manage assets (css, js). The module must first be installed:

```
pip install webassets
```

webassets allows to concatenate your assets and to use almost all of the hype tools of the moment (see the documentation):

- css minifier (*cssmin*, *yuicompressor*, ...)
- css compiler (*less*, *sass*, ...)
- js minifier (*uglifyjs*, *yuicompressor*, *closure*, ...)

Others filters include gzip compression, integration of images in css with *datauri* and more. Webassets also append a version identifier to your asset url to convince browsers to download new versions of your assets when you use far future expires headers.

When using it with Pelican, *webassets* is configured to process assets in the `OUTPUT_PATH/theme` directory. You can use it in your templates with a template tag, for example:

will produce a minified css file with the version identifier:

Another example for javascript:

will produce a minified and gzipped js file:

5.2.8 Example settings

```
# -*- coding: utf-8 -*-
AUTHOR = u'Alexis Métaireau'
SITENAME = u"Alexis' log"
```

```
SITEURL = 'http://blog.notmyidea.org'
TIMEZONE = "Europe/Paris"

GITHUB_URL = 'http://github.com/ametaireau/'
DISQUS_SITENAME = "blog-notmyidea"
PDF_GENERATOR = False
REVERSE_CATEGORY_ORDER = True
LOCALE = "C"
DEFAULT_PAGINATION = 4
DEFAULT_DATE = (2012, 03, 02, 14, 01, 01)

FEED_RSS = 'feeds/all.rss.xml'
CATEGORY_FEED_RSS = 'feeds/%s.rss.xml'

LINKS = (('Biologeek', 'http://biologeek.org'),
         ('Filyb', "http://filyb.info/"),
         ('Libert-fr', "http://www.libert-fr.com"),
         ('N1k0', "http://prendreuncafe.com/blog/"),
         (u'Tarek Ziadé', "http://ziade.org/blog"),
         ('Zubin Mithra', "http://zubin71.wordpress.com/"),)

SOCIAL = (('twitter', 'http://twitter.com/ametaireau'),
          ('lastfm', 'http://lastfm.com/user/akounet'),
          ('github', 'http://github.com/ametaireau'),)

# global metadata to all the contents
DEFAULT_METADATA = (('yeah', 'it is'),)

# static paths will be copied under the same name
STATIC_PATHS = ["pictures", ]

# A list of files to copy from the source to the destination
FILES_TO_COPY = (('extra/robots.txt', 'robots.txt'),)

# foobar will not be used, because it's not in caps. All configuration keys
# have to be in caps
foobar = "barbaz"
```

5.3 How to create themes for Pelican

Pelican uses the great Jinja2 templating engine to generate its HTML output. Jinja2 syntax is really simple. If you want to create your own theme, feel free to take inspiration from the "simple" theme.

5.3.1 Structure

To make your own theme, you must follow the following structure:

```
-- static
|   -- css
|   -- images
-- templates
    -- archives.html    // to display archives
    -- article.html     // processed for each article
    -- author.html      // processed for each author
    -- authors.html     // must list all the authors
    -- categories.html  // must list all the categories
    -- category.html    // processed for each category
    -- index.html       // the index. List all the articles
    -- page.html        // processed for each page
    -- tag.html         // processed for each tag
    -- tags.html        // must list all the tags. Can be a tag cloud.
```

- *static* contains all the static assets, which will be copied to the output *theme/static* folder. I've put the CSS and image folders here, but they are just examples. Put what you need here.
- *templates* contains all the templates that will be used to generate the content. I've just put the mandatory templates here; you can define your own if it helps you keep things organized while creating your theme.

5.3.2 Templates and variables

The idea is to use a simple syntax that you can embed into your HTML pages. This document describes which templates should exist in a theme, and which variables will be passed to each template at generation time.

All templates will receive the variables defined in your settings file, if they are in all-caps. You can access them directly.

Common variables

All of these settings will be available to all templates.

Variable	Description
articles	The list of articles, ordered descending by date All the elements are *Article* objects, so you can access their attributes (e.g. title, summary, author etc.)
dates	The same list of articles, but ordered by date, ascending
tags	A key-value dict containing the tags (the keys) and the list of respective articles (the values)
categories	A key-value dict containing the categories (keys) and the list of respective articles (values)
pages	The list of pages

index.html

This is the home page of your blog, generated at output/index.html.

If pagination is active, subsequent pages will reside in output/index'n'.html.

Variable	Description
articles_paginator	A paginator object for the list of articles
articles_page	The current page of articles
dates_paginator	A paginator object for the article list, ordered by date, ascending.
dates_page	The current page of articles, ordered by date, ascending.
page_name	'index' – useful for pagination links

author.html

This template will be processed for each of the existing authors, with output generated at output/author/*author_name*.html.

If pagination is active, subsequent pages will reside at output/author/*author_name''n*.html.

Variable	Description
author	The name of the author being processed
articles	Articles by this author
dates	Articles by this author, but ordered by date, ascending
articles_paginator	A paginator object for the list of articles
articles_page	The current page of articles
dates_paginator	A paginator object for the article list, ordered by date, ascending.
dates_page	The current page of articles, ordered by date, ascending.
page_name	'author/*author_name*' – useful for pagination links

category.html

This template will be processed for each of the existing categories, with output generated at output/category/*category_name*.html.

If pagination is active, subsequent pages will reside at output/category/*category_name''n*.html.

Variable	Description
category	The name of the category being processed
articles	Articles for this category
dates	Articles for this category, but ordered by date, ascending
articles_paginator	A paginator object for the list of articles
articles_page	The current page of articles
dates_paginator	A paginator object for the list of articles, ordered by date, ascending
dates_page	The current page of articles, ordered by date, ascending
page_name	'category/*category_name*' – useful for pagination links

article.html

This template will be processed for each article, with .html files saved as output/*article_name*.html. Here are the specific variables it gets.

Variable	Description
article	The article object to be displayed
category	The name of the category for the current article

page.html

This template will be processed for each page, with corresponding .html files saved as output/*page_name*.html.

Variable	Description
page	The page object to be displayed. You can access its title, slug, and content.

tag.html

This template will be processed for each tag, with corresponding .html files saved as output/tag/*tag_name*.html.

If pagination is active, subsequent pages will reside at output/tag/*tag_name''n*.html.

Variable	Description
tag	The name of the tag being processed
articles	Articles related to this tag
dates	Articles related to this tag, but ordered by date, ascending
articles_paginator	A paginator object for the list of articles
articles_page	The current page of articles
dates_paginator	A paginator object for the list of articles, ordered by date, ascending
dates_page	The current page of articles, ordered by date, ascending
page_name	'tag/*tag_name*' – useful for pagination links

5.3.3 Feeds

The feed variables changed in 3.0. Each variable now explicitly lists ATOM or RSS in the name. ATOM is still the default. Old themes will need to be updated. Here is a complete list of the feed variables:

```
FEED_ATOM
FEED_RSS
CATEGORY_FEED_ATOM
CATEGORY_FEED_RSS
TAG_FEED_ATOM
TAG_FEED_RSS
TRANSLATION_FEED
```

5.3.4 Inheritance

Since version 3.0, Pelican supports inheritance from the `simple` theme, so you can re-use the `simple` theme templates in your own themes.

If one of the mandatory files in the `templates/` directory of your theme is missing, it will be replaced by the matching template from the `simple` theme. So if the HTML structure of a template in the `simple` theme is right for you, you don't have to write a new template from scratch.

You can also extend templates from the `simple` themes in your own themes by using the `{% extends %}` directive as in the following example:

Example

With this system, it is possible to create a theme with just two files.

base.html

The first file is the `templates/base.html` template:

1. On the first line, we extend the `base.html` template from the `simple` theme, so we don't have to rewrite the entire file.
2. On the third line, we open the `head` block which has already been defined in the `simple` theme.
3. On the fourth line, the function `super()` keeps the content previously inserted in the `head` block.
4. On the fifth line, we append a stylesheet to the page.
5. On the last line, we close the `head` block.

This file will be extended by all the other templates, so the stylesheet will be linked from all pages.

style.css

The second file is the `static/css/style.css` CSS stylesheet:

Download

You can download this example theme `here`.

5.4 Plugins

Since version 3.0, Pelican manages plugins. Plugins are a way to add features to Pelican without having to directly hack Pelican code.

Pelican is shipped with a set of core plugins, but you can easily implement your own (and this page describes how).

5.4.1 How to use plugins

To load plugins, you have to specify them in your settings file. You have two ways to do so. Either by specifying strings with the path to the callables:

```
PLUGINS = ['pelican.plugins.gravatar',]
```

Or by importing them and adding them to the list:

```
from pelican.plugins import gravatar
PLUGINS = [gravatar, ]
```

If your plugins are not in an importable path, you can specify a `PLUGIN_PATH` in the settings:

```
PLUGIN_PATH = "plugins"
PLUGINS = ["list", "of", "plugins"]
```

5.4.2 How to create plugins

Plugins are based on the concept of signals. Pelican sends signals, and plugins subscribe to those signals. The list of signals are defined in a following section.

The only rule to follow for plugins is to define a `register` callable, in which you map the signals to your plugin logic. Let's take a simple example:

```
from pelican import signals

def test(sender):
    print "%s initialized !!" % sender

def register():
    signals.initialized.connect(test)
```

5.4.3 List of signals

Here is the list of currently implemented signals:

Signal	Arguments	Description
initialized	pelican object	
article_generate_context	article_generator, metadata	
article_generator_init	article_generator	invoked in the ArticlesGenerator.__init__
pages_generate_context	pages_generator, metadata	
pages_generator_init	pages_generator	invoked in the PagesGenerator.__init__

The list is currently small, don't hesitate to add signals and make a pull request if you need them!

5.4.4 List of plugins

Not all the list are described here, but a few of them have been extracted from the Pelican core and provided in `pelican.plugins`. They are described here:

Tag cloud

Translation

GitHub activity

This plugin makes use of the `feedparser` library that you'll need to install.

Set the `GITHUB_ACTIVITY_FEED` parameter to your GitHub activity feed. For example, my setting would look like:

```
GITHUB_ACTIVITY_FEED = 'https://github.com/kpanic.atom'
```

On the templates side, you just have to iterate over the `github_activity` variable, as in the example:

```
{% if GITHUB_ACTIVITY_FEED %}
   <div class="social">
           <h2>Github Activity</h2>
           <ul>

           {% for entry in github_activity %}
               <li><b>{{ entry[0] }}</b><br /> {{ entry[1] }}</li>
           {% endfor %}
           </ul>
   </div><!-- /.github_activity -->
{% endif %}
```

`github_activity` is a list of lists. The first element is the title and the second element is the raw HTML from GitHub.

5.5 Pelican internals

This section describe how Pelican works internally. As you'll see, it's quite simple, but a bit of documentation doesn't hurt. :)

You can also find in the *Some history about Pelican* section an excerpt of a report the original author wrote with some software design information.

5.5.1 Overall structure

What Pelican does is take a list of files and process them into some sort of output. Usually, the input files are reStructuredText and Markdown files, and the output is a blog, but both input and output can be anything you want.

The logic is separated into different classes and concepts:

- **Writers** are responsible for writing files: .html files, RSS feeds, and so on. Since those operations are commonly used, the object is created once and then passed to the generators.
- **Readers** are used to read from various formats (Markdown and reStructuredText for now, but the system is extensible). Given a file, they return metadata (author, tags, category, etc.) and content (HTML-formatted).
- **Generators** generate the different outputs. For instance, Pelican comes with `ArticlesGenerator` and `PageGenerator`. Given a configuration, they can do whatever they want. Most of the time, it's generating files from inputs.
- Pelican also uses templates, so it's easy to write your own theme. The syntax is Jinja2 and is very easy to learn, so don't hesitate to jump in and build your own theme.

5.5.2 How to implement a new reader?

Is there an awesome markup language you want to add to Pelican? Well, the only thing you have to do is to create a class with a `read` method that returns HTML content and some metadata.

Take a look at the Markdown reader:

```
class MarkdownReader(Reader):
    enabled = bool(Markdown)

    def read(self, filename):
        """Parse content and metadata of markdown files"""
        text = open(filename)
        md = Markdown(extensions = ['meta', 'codehilite'])
        content = md.convert(text)

        metadata = {}
        for name, value in md.Meta.items():
            if name in _METADATA_FIELDS:
                meta = _METADATA_FIELDS[name](value[0])
            else:
                meta = value[0]
            metadata[name.lower()] = meta
        return content, metadata
```

Simple, isn't it?

If your new reader requires additional Python dependencies, then you should wrap their `import` statements in a `try...except` block. Then inside the reader's class, set the `enabled` class attribute to mark import success or failure. This makes it possible for users to continue using their favourite markup method without needing to install modules for formats they don't use.

5.5.3 How to implement a new generator?

Generators have two important methods. You're not forced to create both; only the existing ones will be called.

- `generate_context`, that is called first, for all the generators. Do whatever you have to do, and update the global context if needed. This context is shared between all generators, and will be passed to the templates.

For instance, the `PageGenerator` `generate_context` method finds all the pages, transforms them into objects, and populates the context with them. Be careful *not* to output anything using this context at this stage, as it is likely to change by the effect of other generators.

- `generate_output` is then called. And guess what is it made for? Oh, generating the output. :) It's here that you may want to look at the context and call the methods of the `writer` object that is passed as the first argument of this function. In the `PageGenerator` example, this method will look at all the pages recorded in the global context and output a file on the disk (using the writer method `write_file`) for each page encountered.

5.6 pelican-themes

5.6.1 Description

`pelican-themes` is a command line tool for managing themes for Pelican.

Usage

pelican-themes [-h] [-l] [-i theme path [theme path ...]]
    [-r theme name [theme name ...]]
    [-s theme path [theme path ...]] [-v] [–version]

Optional arguments:

-h, --help	Show the help an exit
-l, --list	Show the themes already installed
-i theme_path, --install theme_path	One or more themes to install
-r theme_name, --remove theme_name	One or more themes to remove
-s theme_path, --symlink theme_path	Same as "–install", but create a symbolic link instead of copying the theme. Useful for theme development
-v, --verbose	Verbose output
--version	Print the version of this script

5.6.2 Examples

Listing the installed themes

With `pelican-themes`, you can see the available themes by using the `-l` or `--list` option:

In this example, we can see there are three themes available: `notmyidea`, `simple`, and `two-column`.

`two-column` is prefixed with an `@` because this theme is not copied to the Pelican theme path, but is instead just linked to it (see Creating symbolic links for details about creating symbolic links).

Note that you can combine the `--list` option with the `-v` or `--verbose` option to get more verbose output, like this:

Installing themes

You can install one or more themes using the `-i` or `--install` option. This option takes as argument the path(s) of the theme(s) you want to install, and can be combined with the verbose option:

Removing themes

The `pelican-themes` command can also remove themes from the Pelican themes path. The `-r` or `--remove` option takes as argument the name(s) of the theme(s) you want to remove, and can be combined with the `--verbose` option.

Creating symbolic links

`pelican-themes` can also install themes by creating symbolic links instead of copying entire themes into the Pelican themes path.

To symbolically link a theme, you can use the `-s` or `--symlink`, which works exactly as the `--install` option:

In this example, the `two-column` theme is now symbolically linked to the Pelican themes path, so we can use it, but we can also modify it without having to reinstall it after each modification.

This is useful for theme development:

Doing several things at once

The `--install`, `--remove` and `--symlink` option are not mutually exclusive, so you can combine them in the same command line to do more than one operation at time, like this:

In this example, the theme `notmyidea-cms` is replaced by the theme `notmyidea-cms-fr`

5.6.3 See also

- http://docs.notmyidea.org/alexis/pelican/
- `/usr/share/doc/pelican/` if you have installed Pelican using the APT repository

5.7 Import from other blog software

5.7.1 Description

`pelican-import` is a command line tool for converting articles from other software to ReStructuredText. The supported formats are:

- WordPress XML export
- Dotclear export
- RSS/Atom feed

The conversion from HTML to reStructuredText relies on pandoc. For Dotclear, if the source posts are written with Markdown syntax, they will not be converted (as Pelican also supports Markdown).

Dependencies

`pelican-import` has two dependencies not required by the rest of pelican:

- BeautifulSoup
- pandoc

BeatifulSoup can be installed like any other Python package:

```
$ pip install BeautifulSoup
```

For pandoc, install a package for your operating system from the pandoc site.

Usage

pelican-import [-h] [–wpfile] [–dotclear] [–feed] [-o OUTPUT]
    [-m MARKUP][–dir-cat]
    input

Optional arguments

-h, --help	show this help message and exit
--wpfile	Wordpress XML export
--dotclear	Dotclear export
--feed	Feed to parse
-o OUTPUT, --output OUTPUT	Output path
-m MARKUP	Output markup
--dir-cat	Put files in directories with categories name

5.7.2 Examples

for WordPress:

```
$ pelican-import --wpfile -o ~/output ~/posts.xml
```

for Dotclear:

```
$ pelican-import --dotclear -o ~/output ~/backup.txt
```

5.7.3 Tests

To test the module, one can use sample files:

- for Wordpress: http://wpcandy.com/made/the-sample-post-collection
- for Dotclear: http://themes.dotaddict.org/files/public/downloads/lorem-backup.txt

5.8 Frequently Asked Questions (FAQ)

Here is a summary of the frequently asked questions for Pelican.

5.8.1 What's the best way to communicate a problem, question, or suggestion?

If you have a problem, question, or suggestion, please start by striking up a conversation on #pelican on Freenode. Those who don't have an IRC client handy can jump in immediately via IRC webchat. Because of differing time zones, you may not get an immediate response to your question, but please be patient and stay logged into IRC — someone will almost always respond.

If you are unable to resolve your issue or if you have a feature request, please refer to the issue tracker.

5.8.2 How can I help?

There are several ways to help out. First, you can use Pelican and report any suggestions or problems you might have via IRC or the issue tracker.

If you want to contribute, please fork the git repository, create a new feature branch, make your changes, and issue a pull request. Someone will review your changes as soon as possible. Please refer to the *How to Contribute* section for more details.

You can also contribute by creating themes and improving the documentation.

5.8.3 Is it mandatory to have a configuration file?

No, it's not. Configuration files are just an easy way to configure Pelican. For basic operations, it's possible to specify options while invoking Pelican via the command line. See `pelican --help` for more information.

5.8.4 I'm creating my own theme. How do I use Pygments for syntax highlighting?

Pygments adds some classes to the generated content. These classes are used by themes to style code syntax highlighting via CSS. Specifically, you can customize the appearance of your syntax highlighting via the `.codehilite pre` class in your theme's CSS file. To see how various styles can be used to render Django code, for example, you can use the demo on the project website.

5.8.5 How do I create my own theme?

Please refer to *How to create themes for Pelican*.

5.8.6 I want to use Markdown, but I got an error.

Markdown is not a hard dependency for Pelican, so you will need to explicitly install it. You can do so by typing:

```
$ (sudo) pip install markdown
```

In case you don't have pip installed, consider installing it via:

```
$ (sudo) easy_install pip
```

5.8.7 Can I use arbitrary meta-data in my templates?

Yes. For example, to include a modified date in a Markdown post, one could include the following at the top of the article:

```
Modified: 2012-08-08
```

That meta-data can then be accessed in the template:

```
{% if article.modified %}
Last modified: {{ article.modified}}
{% endif %}
```

5.8.8 How do I assign custom templates on a per-page basis?

It's as simple as adding an extra line of metadata to any pages or articles you want to have its own template.

template template_name

Then just make sure to have the template installed in to your theme as `template_name.html`.

5.8.9 What if I want to disable feed generation?

To disable all feed generation set `FEED_ATOM` and `FEED_RSS` to `None` in your settings. Please note `None` and '' are not the same thing. The word `None` should not be surrounded by quotes.

5.8.10 I'm getting a warning about feeds generated without SITEURL being set properly

RSS and Atom feeds require all URLs and links in them to be absolute. In order to properly generate all URLs properly in Pelican you will need to set `SITEURL` to the full path of your blog. When using `make html` and the default Makefile provided by the *pelican-quickstart* bootstrap script to test build your site, it's normal to see this warning since `SITEURL` is deliberately left undefined. If configured properly no other `make` commands should result in this warning.

Feeds are still generated when this warning is displayed but may not validate.

5.8.11 My feeds are broken since I upgraded to Pelican 3.0

Starting in 3.0, some of the FEED setting names were changed to more explicitly refer to the Atom feeds they inherently represent (much like the FEED_RSS setting names). Here is an exact list of the renamed setting names:

```
FEED -> FEED_ATOM
TAG_FEED -> TAG_FEED_ATOM
CATEGORY_FEED -> CATEGORY_FEED_ATOM
```

Older 2.x themes that referenced the old setting names may not link properly. In order to rectify this, please update your theme for compatibility with 3.0+ by changing the relevant values in your template files. For an example of complete feed headers and usage please check out the `simple` theme.

5.9 Tips

Here are some tips about Pelican that you might find useful.

5.9.1 Publishing to GitHub

GitHub comes with an interesting "pages" feature: you can upload things there and it will be available directly from their servers. As Pelican is a static file generator, we can take advantage of this.

User Pages

GitHub allows you to create user pages in the form of `username.github.com`. Whatever is created in the master branch will be published. For this purpose, just the output generated by Pelican needs to pushed to GitHub.

So given a repository containing your articles, just run Pelican over the posts and deploy the master branch to GitHub:

```
$ pelican -s pelican.conf.py ./path/to/posts -o /path/to/output
```

Now add all the files in the output directory generated by Pelican:

```
$ git add /path/to/output/*
$ git commit -am "Your Message"
$ git push origin master
```

Project Pages

For creating Project pages, a branch called `gh-pages` is used for publishing. The excellent ghp-import makes this really easy, which can be installed via:

```
$ pip install ghp-import
```

Then, given a repository containing your articles, you would simply run Pelican and upload the output to GitHub:

```
$ pelican -s pelican.conf.py .
$ ghp-import output
$ git push origin gh-pages
```

And that's it.

If you want, you can put that directly into a post-commit hook, so each time you commit, your blog is up-to-date on GitHub!

Put the following into `.git/hooks/post-commit`:

```
pelican -s pelican.conf.py . && ghp-import output && git push origin gh-pages
```

5.10 How to contribute?

There are many ways to contribute to Pelican. You can enhance the documentation, add missing features, and fix bugs (or just report them).

Don't hesitate to fork and make a pull request on GitHub. When doing so, please create a new feature branch as opposed to making your commits in the master branch.

5.10.1 Setting up the development environment

You're free to set up your development environment any way you like. Here is a way using the virtualenv and virtualenvwrapper tools. If you don't have them, you can install these both of these packages via:

```
$ pip install virtualenvwrapper
```

Virtual environments allow you to work on Python projects which are isolated from one another so you can use different packages (and package versions) with different projects.

To create a virtual environment, use the following syntax:

```
$ mkvirtualenv pelican
```

To clone the Pelican source:

```
$ git clone https://github.com/getpelican/pelican.git src/pelican
```

To install the development dependencies:

```
$ cd src/pelican
$ pip install -r dev_requirements.txt
```

To install Pelican and its dependencies:

```
$ python setup.py develop
```

5.10.2 Running the test suite

Each time you add a feature, there are two things to do regarding tests: checking that the existing tests pass, and adding tests for the new feature or bugfix.

The tests live in "pelican/tests" and you can run them using the "discover" feature of unittest2:

```
$ unit2 discover
```

If you have made changes that affect the output of a Pelican-generated weblog, then you should update the output used by functional tests. To do so, you can use the following two commands:

```
$ LC_ALL="C" pelican -o tests/output/custom/ -s samples/pelican.conf.py \
    samples/content/
$ LC_ALL="C" USER="Dummy Author" pelican -o tests/output/basic/ samples/content/
```

5.10.3 Coding standards

Try to respect what is described in the PEP8 specification when providing patches. This can be eased via the pep8 or flake8 tools, the latter of which in particular will give you some useful hints about ways in which the code/formatting can be improved.

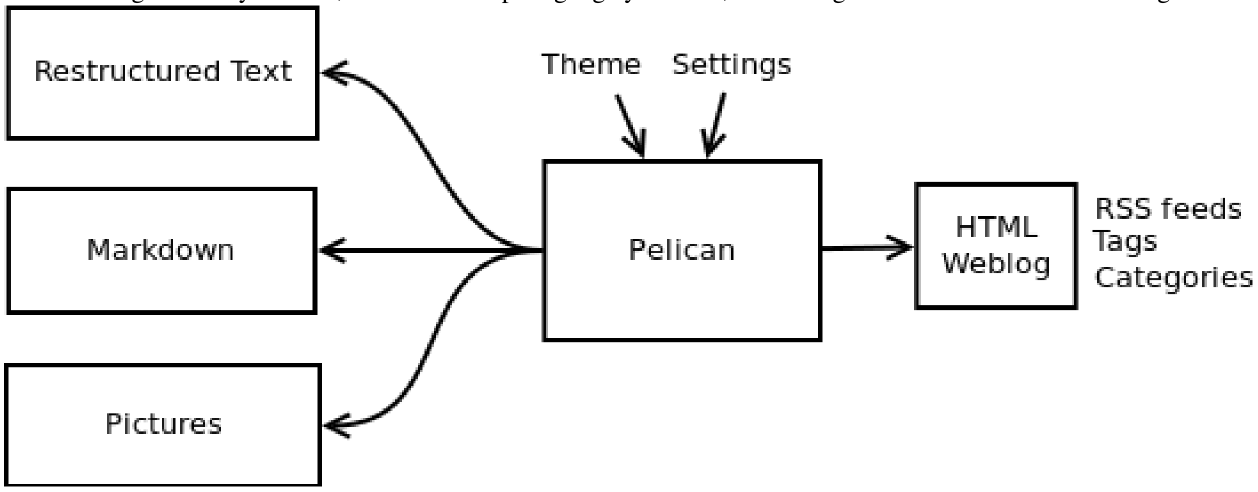
5.11 Some history about Pelican

Warning: This page comes from a report the original author (Alexis Métaireau) wrote right after writing Pelican, in December 2010. The information may not be up-to-date.

Pelican is a simple static blog generator. It parses markup files (Markdown or reStructuredText for now) and generates an HTML folder with all the files in it. I've chosen to use Python to implement Pelican because it seemed to be simple and to fit to my needs. I did not wanted to define a class for each thing, but still wanted to keep my things loosely coupled. It turns out that it was exactly what I wanted. From time to time, thanks to the feedback of some users, it took me a very few time to provide fixes on it. So far, I've re-factored the Pelican code by two times; each time took less than 30 minutes.

5.11.1 Use case

I was previously using WordPress, a solution you can host on a web server to manage your blog. Most of the time, I prefer using markup languages such as Markdown or reStructuredText to type my articles. To do so, I use vim. I think it is important to let the people choose the tool they want to write the articles. In my opinion, a blog manager should just allow you to take any kind of input and transform it to a weblog. That's what Pelican does. You can write your articles using the tool you want, and the markup language you want, and then generate a static HTML weblog.



To be flexible enough, Pelican has template support, so you can easily write your own themes if you want to.

5.11.2 Design process

Pelican came from a need I have. I started by creating a single file application, and I have make it grow to support what it does by now. To start, I wrote a piece of documentation about what I wanted to do. Then, I created the content I wanted to parse (the reStructuredText files) and started experimenting with the code. Pelican was 200 lines long and contained almost ten functions and one class when it was first usable.

I have been facing different problems all over the time and wanted to add features to Pelican while using it. The first change I have done was to add the support of a settings file. It is possible to pass the options to the command line, but can be tedious if there is a lot of them. In the same way, I have added the support of different things over time: Atom feeds, multiple themes, multiple markup support, etc. At some point, it appears that the "only one file" mantra was not good enough for Pelican, so I decided to rework a bit all that, and split this in multiple different files.

I've separated the logic in different classes and concepts:

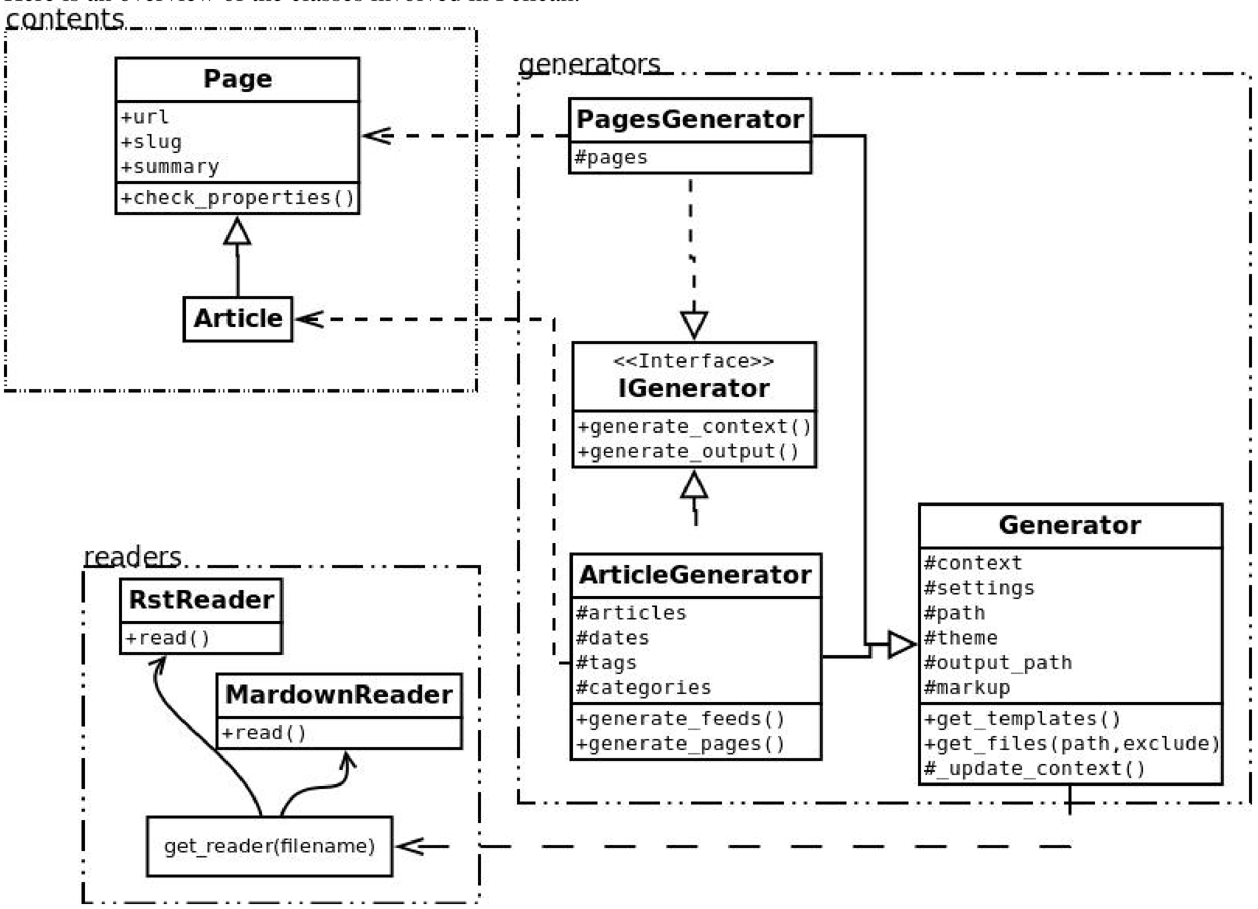
- *writers* are responsible of all the writing process of the files. They are responsible of writing .html files, RSS feeds and so on. Since those operations are commonly used, the object is created once, and then passed to the generators.
- *readers* are used to read from various formats (Markdown and reStructuredText for now, but the system is extensible). Given a file, they return metadata (author, tags, category, etc) and content (HTML formatted).

- *generators* generate the different outputs. For instance, Pelican comes with an ArticlesGenerator and PagesGenerator, into others. Given a configuration, they can do whatever you want them to do. Most of the time it's generating files from inputs (user inputs and files).

I also deal with contents objects. They can be `Articles`, `Pages`, `Quotes`, or whatever you want. They are defined in the `contents.py` module and represent some content to be used by the program.

5.11.3 In more detail

Here is an overview of the classes involved in Pelican.



The interface does not really exist, and I have added it only to clarify the whole picture. I do use duck typing and not interfaces.

Internally, the following process is followed:

- First of all, the command line is parsed, and some content from the user is used to initialize the different generator objects.
- A `context` is created. It contains the settings from the command line and a settings file if provided.
- The `generate_context` method of each generator is called, updating the context.
- The writer is created and given to the `generate_output` method of each generator.

I make two calls because it is important that when the output is generated by the generators, the context will not change. In other words, the first method `generate_context` should modify the context, whereas the second `generate_output` method should not.

Then, it is up to the generators to do what the want, in the `generate_context` and `generate_content` method. Taking the `ArticlesGenerator` class will help to understand some others concepts. Here is what happens when calling the `generate_context` method:

- Read the folder "path", looking for restructured text files, load each of them, and construct a content object (`Article`) with it. To do so, use `Reader` objects.
- Update the `context` with all those articles.

Then, the `generate_content` method uses the `context` and the `writer` to generate the wanted output.

5.12 Release history

5.12.1 3.0 (2012-08-08)

- Refactored the way URLs are handled
- Improved the English documentation
- Fixed packaging using `setuptools` entrypoints
- Added `typogrify` support
- Added a way to disable feed generation
- Added support for `DIRECT_TEMPLATES`
- Allow multiple extensions for content files
- Added LESS support
- Improved the import script
- Added functional tests
- Rsync support in the generated Makefile
- Improved feed support (easily pluggable with Feedburner for instance)
- Added support for `abbr` in reST
- Fixed a bunch of bugs :-)

5.12.2 2.8 (2012-02-28)

- Dotclear importer
- Allow the usage of Markdown extensions
- Themes are now easily extensible
- Don't output pagination information if there is only one page
- Add a page per author, with all their articles
- Improved the test suite
- Made the themes easier to extend
- Removed Skribit support
- Added a `pelican-quickstart` script
- Fixed timezone-related issues

- Added some scripts for Windows support
- Date can be specified in seconds
- Never fail when generating posts (skip and continue)
- Allow the use of future dates
- Support having different timezones per language
- Enhanced the documentation

5.12.3 2.7 (2011-06-11)

- Use `logging` rather than echoing to stdout
- Support custom Jinja filters
- Compatibility with Python 2.5
- Added a theme manager
- Packaged for Debian
- Added draft support

5.12.4 2.6 (2011-03-08)

- Changes in the output directory structure
- Makes templates easier to work with / create
- Added RSS support (was Atom-only)
- Added tag support for the feeds
- Enhance the documentation
- Added another theme (brownstone)
- Added translations
- Added a way to use cleaner URLs with a rewrite url module (or equivalent)
- Added a tag cloud
- Added an autoreloading feature: the blog is automatically regenerated each time a modification is detected
- Translate the documentation into French
- Import a blog from an RSS feed
- Pagination support
- Added Skribit support

5.12.5 2.5 (2010-11-20)

- Import from Wordpress
- Added some new themes (martyalchin / wide-notmyidea)
- First bug report!

- Linkedin support
- Added a FAQ
- Google Analytics support
- Twitter support
- Use relative URLs, not static ones

5.12.6 2.4 (2010-11-06)

- Minor themes changes
- Add Disqus support (so we have comments)
- Another code refactoring
- Added config settings about pages
- Blog entries can also be generated in PDF

5.12.7 2.3 (2010-10-31)

- Markdown support

5.12.8 2.2 (2010-10-30)

- Prettify output
- Manages static pages as well

5.12.9 2.1 (2010-10-30)

- Make notmyidea the default theme

5.12.10 2.0 (2010-10-30)

- Refactoring to be more extensible
- Change into the setting variables

5.12.11 1.2 (2010-09-28)

- Added a debug option
- Added per-category feeds
- Use filesystem to get dates if no metadata is provided
- Add Pygments support

5.12.12 1.1 (2010-08-19)

- First working version