Optimus Documentation

Release 0.8.1

David THENON

January 01, 2017

Contents

Optimus is a static site builder, it produces HTML from templates (through Jinja2) with assets management (through webassets) and i18n support for translations (through Babel).

Read the Optimus documentation on https://optimus.readthedocs.org/

CHAPTER 1

Links

- Read the documentation on Read the docs;
- Download its PyPi package;
- Clone it on its Github repository;

1.1 Contents

1.1.1 Install

You will have to install pip and virtualenv on your system. You should first install pip package then it will be easier to install virtualenv with it, like this :

```
sudo pip install virtualenv
```

It is recommended to install it in a virtualenv environment like this :

```
virtualenv --no-site-packages my_optimus_projects
cd my_optimus_projects
source bin/activate
pip install optimus
```

This way you can work safely on your projects within this environment without any change to your system.

Also you can install it directly on your system without virtualenv, just use pip :

```
sudo pip install optimus
```

Operator system

Optimus has been developed for Linux systems, it works also on Windows and MacOSX but you should have some tasks that will differs from the present documentation.

Asset compressor

Default install comes with yui-compressor as a dependancy to use with webassets because it is a great choice to compress CSS and Javascript assets. But beware that it requires you to have a Java Runtime Environment installed on your system, the OpenJDK is perfectly supported.

Webserver for development

You can install cherrypy, a simple Web server, to see your builded pages :

```
pip install cherrypy
```

Read *Web server* to see how to use it.

Enable i18n support

Then you will have to enable it by adding the Jinja2 i18n extension in your settings :

```
JINJA_EXTENSIONS = (
    ...
    'jinja2.ext.i18n',
    ...
)
```

This is only for a new project manually created, `i18n` project template already installs this for you.

Use Foundation 3

This will require a recent Compass install and thus, a recent Ruby install too. If you encounter problems with this, you can see to rvm to install a more recent Ruby version without touching your system packages.

Just target the correct version when you install the Foundation gem :

```
gem install --version '3.2.5' zurb-foundation
```

Then you should install the plugin to have a project template to create new projects that allready embeds Foundation 3 :

```
pip install optimus_foundation
```

Use Foundation 5

This will also require a recent Compass install and thus, a recent Ruby install too. If you encounter problems with this, you can see to rvm to install a more recent Ruby version without touching your system packages.

Then just install the plugin to have a project template to create new projects that allready embeds Foundation 5 :

```
pip install optimus_foundation_5
```

1.1.2 Basics

Optimus is usable with a command line tool to build pages, create new projects or enter in a watch mode that automatically rebuilds pages when their templates has been changed.

It works a little bit like Django as you create a project with a **settings** file containing all useful global settings to build your pages and manage your assets.

Settings

This is where your environment configuration resides, generally the `settings.py` is the default settings used in development, and the `prod_settings.py` file is used for a production environment that it inherits from the default settings and only sets a `DEBUG = False` to avoid the debug mode and minify the assets.

Optimus command line actions always accept a `settings` option to specify a settings file, by default this is the `settings.py` that is used but if you want to use another settings file like `prod_settings.py` you have to specify it in command line like a Python path :

```
optimus-cli [ACTION] --settings=prod_settings
```

If you just want to use the default settings, you don't need to specify it with `settings` option.

Below is a list of all available settings, but not all are created in the settings file when you create a new project with Optimus, only the useful ones. Optionnal settings that are undefined will be set with a default value. When the default value is not defined in the list, you can assume than they are empty.

DEBUG When actived (`True`) Optimus will not try to compress asset bundles. It is useful to avoid to re-compress them after any changes, this is the preferred method when developping your templates and CSS, this is why it is the default behavior in the default settings file. Disable it for production settings (this is already done in the production settings file provided in project templates. You can also access this variable from your templates if needed.

PROJECT_DIR Absolute path to the project directory. The settings files provided in project templates already fills them automatically, you should not need to edit it.

SITE_NAME The project name to use in your templates.

SITE_DOMAIN The project hostname (without http protocol prefixe) to use in your templates.

SOURCES_DIR Absolute path to the project sources (templates, assets, etc..) directory.

TEMPLATES_DIR Absolute path to the project templates directory.

PUBLISH_DIR Absolute path to the directory where to publish pages and assets. Don't use the same path for different settings file.

STATIC_DIR Absolute path where will be moved all the static files (from the sources), usually this is a directory in the `PUBLISH_DIR`

LOCALES_DIR Absolute path to the i18n translation catalogs directories.

WEBASSETS_CACHE The directory where webassets will store his cache. You can set this to `False` to not use the cache, or set it to True to use the default directory from webassets.

LANGUAGE_CODE Language locale name to use as the default for Pages that don't define it, see http://www.i18nguy.com/unicode/language-identifiers.html

LANGUAGES A list of locale name for all available languages to manage with PO files. Remember to add it the locale name for the default language from `LANGUAGE_CODE`.

Sample :

```
LANGUAGES = (LANGUAGE_CODE, 'fr_FR')
```

This will add the default language and French to the known languages to manage.

Sometime it is also needed to have a label for these languages or some other associated parameters, so your languages entries can be tuples but their first item **must** be the locale name. Here is a sample :

```
LANGUAGES = (
    (LANGUAGE_CODE, "International"),
    ('fr_FR', "France"),
)
```

Note that Optimus didn't care about other items in tuples of languages entries, you can add everything you want. But take care that Optimus will allways assume the first item is the locale name it needs.

STATIC_URL The static url to use in templates and with webassets. This can be a full URL like `http://`, a relative path or an absolute path.

RST_PARSER_SETTINGS ReSTructuredText parser settings to use when building a RST document. This is only useful if you use RST documents in your pages.

Default value is :

```
RST_PARSER_SETTINGS = {
    'initial_header_level': 3,
    'file_insertion_enabled': True,
    'raw_enabled': False,
    'footnote_references': 'superscript',
    'doctitle_xform': False,
}
```

EXTRA_BUNDLES This setting name is deprecated and will be removed in futur release. This was the previous name for `BUNDLES` setting.

BUNDLES Custom bundles to use for managing assets.

Sample :

```
BUNDLES = {
    'my_css_bundle': Bundle(
        'css/app.css',
        filters='yui_css',
        output='css/app.min.css'
    ),
    'my_js_bundle': Bundle(
        'js/app.js',
        filters='yui_js',
        output='js/app.min.js'
    ),
}
```

ENABLED_BUNDLES Key names of enabled bundles to use, by default all known bundles (from setting `BUNDLES`) are enabled. If you don't want to enable them all, just define it with a list of bundle names to enable.

FILES_TO_SYNC Sources files or directories to synchronize within the published static directory. This is usually used to put on some assets in the static directory like images that don't need to be compressed with assets bundles.

Note that you should be carefull to not conflict with files targeted by webassets bundles.

JINJA_EXTENSIONS Comment, uncomment or add new extension path to use with Jinja here.

Default value is :

```
JINJA_EXTENSIONS = (
    'jinja2.ext.i18n',
)
```

Note that you don't need to manually define the webassets extension if you use it, it is automatically appended within the build process if it detects bundles.

PAGES_MAP Python path to the file that contains pages map, this is relative to your project, default value is `pages`, meaning this will search for `pages.py` file in your project directory.

I18N_EXTRACT_MAP Map for translation rules extraction with Babel.

Default value is :

```
I18N_EXTRACT_MAP = (
    ('pages.py', 'python'),
    ('*settings.py', 'python'),
    ('**/templates/**.html', 'jinja2'),
)
```

So the default behavior is only to search for translations in template sources, `pages.py` and all common settings files.

I18N_EXTRACT_SOURCES List of path to search for translation to extract. In these paths, a scan will be done using the rules from `I18N_EXTRACT_MAP`.

Default value is :

```
I18N_EXTRACT_SOURCES = (
    PROJECT_DIR,
)
```

So it will search recursively in the project directory.

I18N_EXTRACT_OPTIONS Options for translation rules extraction with Babel.

Default value is :

```
I18N_EXTRACT_OPTIONS = {
    '**/templates/**.html': {
        'extensions': 'webassets.ext.jinja2.AssetsExtension',
        'encoding': 'utf-8'
    }
}
```

Templates

The templates are rendered to pages using template engine Jinja2.

For each template the default context variables are :

- `debug` : A boolean, his value comes from `settings.DEBUG`;
- `SITE` : A dict containing some variables from the settings;
  - `name` : the value from `settings.SITE_NAME`;
  - `domain` : the value from `settings.SITE_DOMAIN`;
  - `web_url` : the value from `settings.SITE_DOMAIN` prefixed by *http://*;
- `STATIC_URL` : A string, containing the value from `settings.STATIC_URL`;

Read the Jinja2 documentation for more details on the available template markups.

Assets

You can simply put your assets where you want in the `sources` directory and add your assets directories in `settings.FILES_TO_SYNC`, they will be copied to your build directory.

But with Optimus this is only required for *real* static assets like images. For CSS and Javascript you should manage them with webassets that is already installed with Optimus.

With webassets you manage your assets as **packages** named `Bundle`, like a bundle for your main CSS, another for your IE CSS hacks/patchs and another for your Javascripts files. You will have to register your custom bundles in `settings.BUNDLES` and enable them in `settings.ENABLED_BUNDLES`.

The benefit of webassets is that it can pre and post process all your assets. This is usually used to *minify* and pack multiple files in one final file. Read the webassets documentation for more details how to use this and to manage bundle assets in your templates.

Pages

The pages to build are registred in a `pages.py` file in your project, it must contains a `PAGES` variable that is a list containing `optimus.builder.pages.PageViewBase` instances.

A default project created from the `init` (*Create a project*) command is already shipped with a `pages.py` containing some samples pages, you can change them, inherit them or add another to build various pages.

Page context

Default `PageViewBase` instance adds some variables to its template context (*Templates*) :

- **page_title** that contains the value of `PageViewBase.title` attribute;
- **page_destination** that contains the value of `PageViewBase.destination` attribute;
- **page_relative_position** that contains the relative path position from the destination file to the root of the publish directory;
- **page_lang** that contains the value of `PageViewBase.page_lang` attribute;
- **page_template_name** that contains the value of `PageViewBase.template_name` attribute;

See `optimus.builder.pages` to see more detail on how it works.

Defining your pages

There are three required arguments for a `PageViewBase` object :

title The title of your page, can be anything you want, it's just a context variable that you can use in your templates.

destination Destination file path where the page will be builded, the path is relative to the setting `PUBLISH_DIR`. You can use multiple subdirectory levels if needed, the builder will create them if it does not allready exists.

template_name File path for the template to use, the path is relative to the setting `TEMPLATES_DIR`.

The short way is like so :

```
from optimus.builder.pages import PageViewBase
# Enabled pages to build
PAGES = [
    PageViewBase(title="My page", template_name="mypage.html", destination="mypage.html"),
]
```

But it is more likely you need to build more than one pages and generally you want to share some attributes like templates or title. So instead of directly using `PageViewBase`, you should make your own page object like this :

```
from optimus.builder.pages import PageViewBase

class MyBasePage(PageViewBase):
    title = "My base page"
    template_name = "mypage.html"

# Enabled pages to build
PAGES = [
    MyBasePage(title="My index", destination="index.html"),
    MyBasePage(title="My Foo page", destination="foo.html"),
    MyBasePage(title="My Bar page", destination="bar.html"),
]
```

Extending PageViewBase

You can override some methods to add logic or change some behaviors in your `PageViewBase` object.

PageViewBase.get_title Set the `page_title` context variable.

PageViewBase.get_destination Set the `page_destination` context variable.

PageViewBase.get_relative_position Set the `page_relative_position` context variable.

PageViewBase.get_lang Set the `page_lang` context variable.

PageViewBase.get_template_name Set the `page_template_name` context variable.

PageViewBase.get_context Set the context page to add variables to expose in the templates. The method does not attempt any argument and return the context.

To add a new variable `foo` in your context you may do it like this :

```
    class MyPage(PageViewBase):
        title = "My page"
        template_name = "mypage.html"
        destination = "mypage.html"

        def get_context(self):
            # This line set the default context from PageViewBase
            super(MyPage, self).get_context()
            # Add your new variables here
            self.context.update({
                'foo': 'bar',
            })
            return self.context
```

Translations

Marked strings with the `{% trans %}` template tag in your templates (see Jinja2 template documentation) will be translated from the page locale name and its associated translation catalog. They will be extracted and stored in catalog files where you will have to fill the translations. Then compile your catalog files and then, the page building will replace strings with the translation accordingly to the page language.

The recommended way is to use the Optimus command `po` see this in *Managing translations*.

Pages language

By default, Pages use a default locale language that is *en_US*, for each language you will need to make a page view with the wanted language. You can specify it in the **lang** page attribute, or in a `lang` argument when you instanciate your `PageViewBase`.

Managing translation catalog with the raw way

The *raw* way is to directly use Babel command line tool, you will have many more option to manage your catalogs but you will have to use many different commands and paths.

Before building your internationalized Pages, you will have to create a messages catalog for each needed language. Put all your `{% trans %}` tags in your templates, then make a catalog from the extracted string.

To correctly extract all your strings to translate, Babel will need some rules to know what and where it should search. This is done in a Babel mapping file, generally as a `babel.cfg` in the root directory of your project.

At least, you will need the Jinja2 integration rule :

```
[jinja2: sources/templates/**.html]
encoding = utf-8
extensions = webassets.ext.jinja2.AssetsExtension
```

The last line is needed if you use webassets tags `{% assets %}...{% endassets %}` in your templates, otherwise the extraction will fail. See the Jinja2 integration documentation for more details.

Extracting first the reference POT file :

```
pybabel extract -F babel.cfg -o locale/messages.pot .
```

Initialize the language files (repeat this for each needed language with his correct locale key) :

```
pybabel init -l en_US -d locale -i locale/messages.pot
```

Compile all your language files :

```
pybabel compile -f -d locale
```

Update them when you make changes in your template strings (after this, you'll need to re-compile them) :

```
pybabel update -l en_US -d locale -i locale/messages.pot
```

1.1.3 Usage

You can use Optimus from the command line tool `optimus-cli`. A global help is available with :

```
optimus-cli help
```

Or specific command action help with :

```
optimus-cli help action_name
```

There is also a common command argument `--settings` (apart from the `init` command) that is useful to define the settings files to use. It appends a Python path to the settings file. For common usage you just have to give the filename without the `.py` extension, otherwise you will get an error message.

Create a project

At least you will give a name for the new project. Be aware that it must a valid Python module name, so only with alphanumeric characters and _. No spaces, no dots, etc.. :

```
optimus-cli init my_project
```

It will create project directory and fill it with basic content. But Optimus can use project templates to create project more useful :

- `basic` : This is the default one, included in Optimus, you don't have to specify anything to use it;
- `i18n` : The i18n sample, included in Optimus. All needed stuff to enable i18n support are installed;
- `optimus_foundation` : Optimus-foundation that create a new project including all Foundation **3** stuff, you will have to install it before (see *Use Foundation 3*);
- `optimus_foundation_5` : Optimus-foundation-5 that create a new project including all Foundation **5** stuff, you will have to install it before (see *Use Foundation 5*);

To create a new project with the I18n sample, you will have to do something like :

```
optimus-cli init my_project -t i18n
```

To create a new Foundation project with Optimus-foundation-5 plugin :

```
optimus-cli init my_project -t optimus_foundation_5
```

Building

Configure your settings if needed, then your Pages to build and finally launch Optimus to build them :

```
optimus-cli build
```

Managing translations

Optimus can manage your translations for the known languages of your project. This is done in the setting `LANGUAGES` where you define a list of locale names, each of which will have a translation catalogs after you initialize them. By default, this settings is only filled with the default locale defined in the settings `LANGUAGE_CODE`. This is your responsability to fill the setting `LANGUAGES` with valid locale names.

Assuming you want to add French translations, you will have to add this setting :

```
# A list of locale name for all available languages to manage with PO files
LANGUAGES = (LANGUAGE_CODE, 'fr_FR')
```

Note the first item that also adds the locale name from the default language from the setting `LANGUAGE_CODE`.

Then you will need to flag the strings to translate in your templates with the `{% trans %}` template tag from Jinja2 (see Jinja2 template documentation for more details) like this :

```
<html>
<body>
    <h1>{% trans %}Hello world{% endtrans %}</h1>
</body>
</html>
```

And finally manage your translation catalogs, see below.

Initialize

On a new project you have to initialize the catalog template (the source used to create or update translation catalogs, represented by a `*.POT` file in your locales directory) :

```
optimus-cli po --init
```

This will extract translation strings from your templates (and other files in your sources directory if needed) and put them in catalog templates, then after translation catalogs will be created from the template for each knowed languages.

This command is safe for existing translations, if a translation catalogs allready exists, it will not be overwrited. Only non existing translation catalogs will be created.

Now open your catalog files (`*.PO`) edit them to fill the translations for your languages, then compile them (see *Compilation*).

Update

If you do some changes on translations in your templates, like add new translation strings, modify or remove some, you have to update your catalogs to adapt to this changes :

```
optimus-cli po --update
```

This will extract again your translation strings, update the catalog template then update your translation catalogs. After that you will have to re-compile them (see *Compilation*).

Compilation

Catalog files (`*.PO`) are not usable for page building, you will have to compile them first, this is done with the command line :

```
optimus-cli po --compile
```

It will compile the catalog file to `*.MO` files, this way Optimus can use your translations. Remember that when you do updates on catalog files you will have to re-compile them each time, this is not automatic.

Note that also when you edit your translation catalogs to change some translations, you will have to re-compile them.

Watch mode

Use the `watch` command action to automatically rebuild files at each change in your sources :

```
optimus-cli watch
```

This will launch a process that will watch for changes and rebuild pages if needed. For changes on templates, the watch mode will only rebuild pages that uses the changed templates. Also if it detects that the publish directory (from the setting `PUBLISH_DIR`) does not exists, it will automatically performs a first build.

To stop the watcher process, just use the common keyboard combo `CTRL+C`.

This is useful in development, but note that the watcher is limited to watch only for templates and assets changes.

Watch mode will not detect if :

- You change some things in your Page views, your settings or your RST files;
- You add new static files;

- You make some changes in your translation files (`*.pot` and `*.po`);

For theses cases you will have to stop the watcher, manually rebuild with `build` command or Babel tool (for translations only) then relaunch the watcher.

Web server

You can launch a simple web server to publish your builded content, **it's not intended to be used in production**, only for debugging your work. This command action is only available if you already have installed **cherrypy**, see the *Install* document about this.

The hostname argument is required and it should at least contain the port (like '80'), the default address will be "127.0.0.1" if you don't give it.

To launch the webserver binded on your local IP on port 8001 to publish your project from the default settings, do this :

```
optimus-cli runserver 0.0.0.0:8001
```

Also you can bind it on localhost on port 8080 with the production settings :

```
optimus-cli runserver localhost:8080 --settings=prod_settings
```

The settings are used to know the publish directory to expose.

Note that the server does not build anything, it only expose the publish directory to publish the builded page and static files it contains. You should launch the *Watch mode* in parallel.

1.1.4 Changelog

Version 0.8.1 - Unreleased

- Validated working with `CherryPy==8.7.0`, so remove every occurences about 3.x.x version;
- Better README/Doc index/Package short description;

Version 0.8.0 - 2016/12/31

- Include `html5writer.py` taken from `rstview` and so remove dependency to `rstview`, close #19;
- Move changelog to its own file, updated documentation Makefile, added dev requirements;
- Use `sphinx_rtd_theme` in documentation if available;
- Improved watcher logging output a little bit so it reveals changed file when detected without to use the debug level;
- Do not enable anymore `runserver` command to installed CherryPy, instead raise a better error message explanation;

Version 0.7.2 - 2016/05/05

Minor update that modify 'settings' and 'pages' modules import so exception is raised to ease debugging.

Version 0.7.1 - 2015/06/14

Dummy release just to update documentation about forgotted changelog.

Version 0.7.0 - 2015/06/14

- Upgraded dependancy to watchdog==0.8.3 to try to fix a problem with watch mode on OSX;
- Fixed doc;
- Changed module imports to have distinct error name for page and settings import errors;
- Changed message error for module loading to be more helpful;

Version 0.6.9

- Fix a bug with bad signature for `po` command;
- Moving script name from **optimus** to **optimus-cli** because this was causing issues with `setup.entry_points` usage and buildout;

Version 0.6.8.1

Update Argh dependancy to `>= 0.24.1`.

Version 0.6.8

Re-use a fixed version for **argh** because the 0.24 version has incompatible backward issues.

Version 0.6.7.1

Fix dependancies syntax in setup.py that was causing issues during installation.

Version 0.6.7

- Remove CherryPy dependancy from setup.py, add an install note about this;
- Update documentation;

Version 0.6.6

Upgrade to yuicompressor 2.4.8

Version 0.6.5

Updating doc, in setup.py use 'entry_points' instead of 'scripts'

Version 0.6.4

- Fixing update method in po command to update the POT file;
- Add I18N_EXTRACT_SOURCES setting and use it in extraction method, bumping version;
- Add new behavior for settings.LANGUAGES to permit tuples instead of simple locale name;

Version 0.6.1

- Setting name `EXTRA_BUNDLES` is deprecated and **will be removed in a futur release**. In project settings rename it to `BUNDLES`;
- Remove `optimus.builder.assets.COMMON_BUNDLES`, this was containing default bundles that was not really useful. If your project used them, you will have errors on page building about missing bundles, you can recover them in your `settings.BUNDLES` from :

```
COMMON_BUNDLES = {
    'css_screen_common': Bundle(
        'css/screen.css',
        filters='yui_css',
        output='css/screen.min.css'
    ),
    'css_ie_common': Bundle(
        'css/ie.css',
        filters='yui_css',
        output='css/ie.min.css'
    ),
    'js_ie_common': Bundle(
        'js/modernizr.custom.js',
        'js/respond.src.js',
        filters='yui_js',
        output='js/ie.min.js'
    ),
    'js_jquery': Bundle(
        'js/jquery/jquery-1.7.1.js',
        filters='yui_js',
        output='js/jquery.min.js'
    ),
}
```

Version 0.6 - 2013/12/16

- Add new command `po` to automatically manage translations files;
- Add better error messages for some command line options;
- Add a required settings list that is checked when loading settings file to avoid error on missing settings;
- Add default values to un-required settings so the settings file is more clean and short with only needed settings;
- Now Babel, cherrypy and yui-compressor are required dependancies;
- The previous commande line tool name `optimus-cli` has been chaned to a more shorter name `optimus`;
- New settings have been added to manage languages and translations with the new command `po`;
- Settings files have been simplified, making some settings optionnal to have a more clean and short settings files;

- `watch` command options : automatically perform the first build when the build directory does not exits to avoid errors with the watcher;
- `init` command options : `--name` has moved to a positionnal argument;
- Project templates : Removed requirements.txt for pip since the `setup.py` contains all needed stuff;
- Project templates : Renamed “sample” to “basic” and “sample_i18n” to “i18n”. Also add aliases for them, so you just have to use their names and not anymore their full Python paths;
- Project templates : Changing to better templates with assets, SCSS sources and Compass config;

CHAPTER 2

Indices and tables

- genindex
- search