Jules Documentation

Release 0.3.0

Calvin Spealman

July 22, 2013

CONTENTS

Jules is a static blog generator named after a victorian-ish era literary or intellectual character, because that's a trendy thing to do.

The design is flexible and plugin-oriented. Much of the built-in functionality is available through a set of plugins, which creates an architecture very prone to adaptation and customization. One of the major goals (though not yet reached) is template impartiality.

Today, Jules is a very capable little static website generator you may find useful for your personal, project, or organization site.

Jules is maintained by Calvin Spealman (AKA @ironfroggy)

You can install Jules easily with

```
pip install jules
```

Which will install the latest version from PyPI.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Getting Started

First, you'll need to install Jules. You can do so with a simple `pip install Jules`, or you can checkout a copy directly from the GitHub repository. You can quickly begin your first Jules website with the `init` subcommand.

```
jules init my-new-site
cd my-new-site
jules build
jules serve
```

Which will produce and empty site with all the parts a Jules project needs. You can view the site, running with the `jules serve` command, by visiting *http://localhost:8000/* on your local machine.

```
my-new-site/
-- site.yaml
-- contents/
|   -- index.yaml
|   -- first-post.yaml
|   -- first-post.rst
-- static/
|   -- css/
|   |   -- site.less
|   -- img/
|   -- js/
-- templates/
    -- site_base.j2
    -- tag.j2
    -- post.j2
    -- index.j2
```

We can break down the files produced and what all the parts do.

- ***site.yaml*** Provides global configuration for the site
- ***contents/*** All your Restructured Text content goes here; Your blog posts and pages.
- ***templates/*** Jinja2 templates defining your layout are found here.
- ***static/*** Javascript, images, and CSS and Less stylesheets are all located under `static/`

1.1.1 Site Configuration

Your *site.yaml* will contain a basic configuration and a few values will obviously need customized.

```
title: "My New Site"
subtitle: "A Website For Me!"
google_analytics_id: "UA-XXXXXX-XX"
domain: "www.example.com"
default_author: "You <you@example.com>"
packs:
- "jules:html5boilerplate"
- "jules:lesscss"
- "jules:atom"
- "jules:pygments"
- "dir:templates"
- "dir:static"
- "dir:contents"
bundle_defaults:
    output_ext: ""
    render: jinja2
collections:
    tags:
        key_pattern: "{value}"
        match:
            status: published
        group_by:
            in: tags
        meta:
            render: jinja2
            template: tag.j2
```

While most of these are self-explainatory, we'll walk through.

```
title: "My New Site"
subtitle: "A Website For Me!"
google_analytics_id: "UA-XXXXXX-XX"
domain: "www.example.com"
default_author: "You <you@example.com>"
```

These basic fields should be the most obvious. These fields, like everything in `site.yaml`, will be available in templates through the `config` variable.

```
bundle_defaults:
    output_ext: ""
    render: jinja2
```

We set a few defaults for pages that will be rendered on our site. We want to render with no extension, for clean URLs, and we want to render with jinja2 templates.

```
packs:
- "jules:html5boilerplate"
- "jules:lesscss"
- "jules:atom"
- "jules:pygments"
- "dir:templates"
- "dir:static"
- "dir:contents"
```

We configure our site with a list of *packs*. Every pack is a directory containing content, assets, or templates, which are all combined into your final site. This allows Jules to offer bits of functionality and shortcuts in self-contained pieces for you to pick or skip, as they suit your needs.

In this default site, we have three packs from Jules offering a basic layout based on the excellent HTML5 Boilerplate

project, the assets needed to use LESS CSS for easier site styling, and the atom pack to provide a template to generate Atom feeds.

The other three packs are the directories you'll find in the site folder, separated for you into templates, static files, and content. These are used in the default starter site, but you can define any packs however you want to split up your site's input.

```
collections:
    tags:
        key_pattern: "{value}"
        match:
            status: published
        group_by:
            in: tags
        meta:
            render: jinja2
            template: tag.j2
```

Collections allow us to group pages on the site with some common attribute, sharing tags in this case. The collection of pages is itself a page we can provide a tag for to render into the site, so we'll have pages for every tag found.

We could group pages by other factors, such as posts made in a series or separate blogs on a single installation.

1.1.2 Contents

The part you'll spend most of your time in is the `contents/` directory.

The files here define all the pages, blog posts, and other types of content you'll use on your site. Typically, a page consists of a configuration file (YAML) and a content file (ReST).

In the example site provided, there is an `index.yaml` configuring the front page, and a pair of files `first-post.yaml` and `first-post.rst` defining a sample blog post. These are grouped into the bundles `index` and `first-post`.

```
template: index.j2
```

In the site configuration we set bundle defaults to render with Jinja2, so all we need to specify for the front page is the name of the template.

A bigger example is the example blog post, which is actually two files. The first configures the page (`first-post.yaml`) and the second contains the contents of the post (`first-post.rst`). The contents are in restructured text, which is a great mark up language to write plain-text which can be converted into a number of presentation formats.

Other content formats can be added easily by the plugin architecture, and more will be included out of the box soon.

```
title: First Post
status: published
publish_time: !!timestamp '2012-06-23 10:00:00'
render: jinja2
template: post.j2
tags:
- test
```

The post is configured with a title, a publication status and time, and the same render and template directives shared by the front page. The post also has a list of tags, and the templates provided will generate a list of all the published tags and build index pages for all of them.

Now, to see our site we just need to run `jules build` from inside the site directory. Jules will look for the `site.yaml` and then load all the configured packs, parse your content, collect your posts and tags, and render the site to `_build`.

Finally, if you want to see the site in action, just run the serve command after building.

```
jules serve
```

And direct your browser to http://localhost:8000/ to see it in action.

When you're ready to deploy, your complete website is sitting in `./_build` waiting to be copied to your webserver.

1.2 Jules API

class `jules.`**`JulesEngine`**(*src_path*)

`add_bundles`(*bundles*, *replace=False*)
Add additional bundles into the engine, mapping key->bundle.

`find_bundles`()
Find all bundles in the input directories, load them, and prepare them.

`get_bundle`(**args*, ***kwargs*)
With the same parameters as get_bundles_by() find exactly one bundle, and raise ValueError if 0 or more than 1 are found.

`get_bundles_by`(**args*, ***kwargs*)
Find bundles in the engine filtered and ordered as needed.

order_key: The name of a meta field to order the results by order: 'asc' or 'desc' (default: 'asc') limit: The number of resuls to return (default: unlimited)

Any additional keyword arguments are taken as meta field values which bundles must match in order to be matched.

For example, to find all published bundles and give the most recent first:

engine.get_bundles_by('updated_time', 'desc', status='published')

`get_template`(*name*)
Load one template by name.

`load_config`()
Populates engine.config with the site configuration.

`load_plugins`(*ns='jules.plugins'*)
Load engine plugins, and sort them by their plugin_order attribute.

`middleware`(*method*, **args*, ***kwargs*)
Call each loaded plugin with the same method, if it exists.

`pipeline`(*method*, *first*, **args*, ***kwargs*)
Call each loaded plugin with the same method, if it exists, passing the return value of each as the first argument of the next.

`prepare`()
Prepare a site by loading configuration, plugins, packs, and bundles.

`prepare_bundles`()
Prepare the bundles, allow plugins to process them.

render_site(*output_dir*)
Render all bundles to the output directory.

walk_bundles()
Iterate over (key, bundle) pairs in the engine, continuing to yield new bundles if they are added to the engine during the process of walking over the bundles.

class jules.**Bundle**(*key*, *defaults=None*)
Each bundle is a collection of input files, properties, and meta data.

add(*input_dir*, *directory*, *filename*)
Add a single file to the bundle.

by_ext(*ext*)
Find one file in the bundle with the given extension, or return None

content = None

get_bundles()

meta
Bundle meta data loaded from a YAML file in the bundle.

parts

prepare(*engine*)
Prepare the bundle for the engine.

recent
The updated, published, or created time. The first to exist is given.

render(*engine*, *output_dir*)
Render the bundle to the output. "Render" can mean rendering a Jinja2 template or simply copying files, depending on the bundle and configuration.

url()
Find the url the bundle will be rendered to.

write_meta()
Writes any changes in the meta data back to the YAML file.

1.3 Template API

Within your templates, you'll have access to a few things that let you access both the current page being rendered and other pages, as well as information about the site itself.

1.3.1 bundle

The current page being rendered.

Bundle.**recent**()
The updated, published, or created time. The first to exist is given.

Bundle.**url**()
Find the url the bundle will be rendered to.

meta

The configured metadata for the current bundle is available easily.

- `meta.title`
- `meta.publish_time`
- `meta.created_time`
- `meta.updated_time`
- `meta.status`
- `meta.tags`

This is all the data from the bundle's YAML file.

1.3.2 engine

The Jules "Engine" usde to process and coordinate the website rendering is available for access to a lot of useful facilities.

`JulesEngine.`**`get_bundles_by`**`(`**args*, ***kwargs*`)`
Find bundles in the engine filtered and ordered as needed.

order_key: The name of a meta field to order the results by order: 'asc' or 'desc' (default: 'asc') limit: The number of resuls to return (default: unlimited)

Any additional keyword arguments are taken as meta field values which bundles must match in order to be matched.

For example, to find all published bundles and give the most recent first:

engine.get_bundles_by('updated_time', 'desc', status='published')

`JulesEngine.`**`get_bundle`**`(`**args*, ***kwargs*`)`
With the same parameters as get_bundles_by() find exactly one bundle, and raise ValueError if 0 or more than 1 are found.

Head to the *API documentation* for everything available here.

1.3.3 config

The parsed `site.yaml` configuration is available from any template.

1.3.4 bundles

All of the bundles on the site are available through the `bundles` variable.

1.4 Roadmap

1.4.1 Plans

Remove bundle status in favor of using source control branches for drafts

Instead of trying to maintain status of items to keep unpublished things from hitting the listing pages and feeds, just create a branch and write there. Merge back to master to publish. Jules will assume you keep your site in a good version control system, like Git or Mercurial.

Possibly this might include optional commands to manage the post branches for you.

Change atom feeds to be explicit

Rather than add every published page to feeds, I'd like feed items to be added as an explicit action. This allows a few useful features.

- Published posts that aren't in feeds. Good for archive and index pages
- Can re-post something to a feed when it gets an update
- Can control better what contents go into a feed

1.4.2 Changelog

0.2.0.1

- Fixed generation of per-collection (tag listings, etc.) Atom feeds

0.2

- Added iso8601 filter for Atom complient datetime formats
- Rendered output is UTF8 allows entities defined in ReST
- When adding bundles, check for duplicates (conflicts between real and implied bundles)
- Updated the starter site to improve formatting
- Allow bundle configurations to be defined for collection/tag pages, as long as the bundle key matches.
- Added tox tests to ensure packaging, install, site init, and site building work correctly on 2.7 and 3.2

0.1.1

- Documentation improvements

0.1

- Initial release

CHAPTER

TWO

OVERVIEW

Jules one or more input directories (called *packs*), combines groups of files (*bundles*) which share their base name. For example, "projects.j2" as a template and "projects.yaml" as data, may be one bundle. These, after processing and allowing plugins a chance to extend Jules' abilities, are rendered into yhour final site.

An example layout:

Packs	Bundles	Files
HTML5 Boiler Pack	base js/jquery.min	base.j2 lib/jquery.min.js
LESS CSS Pack	js/less.min	js/less.min.js
Blog Starter Your site pack site_base	index site_base.j2	index.j2 index.yaml

which might render into the a site like:

```
/
-- index.html
-- js/
    -- jquery.min.js
    -- less.min.js
```

Get started making sites easily today.