Flask-Themes2 Documentation

Release 0.1.2

Christopher "plausibility" Carter, Drew Lustro, Matthew "LeafStor

May 28, 2014

Flask-Themes2 makes it easy for your application to support a wide range of appearances.

CHAPTER 1

Writing Themes

A theme is simply a folder containing static media (like CSS files, images, and JavaScript) and Jinja2 templates, with some metadata. A theme folder should look something like this:

```
my_theme
-- info.json
-- license.txt
-- static
|   -- style.css
-- templates
    -- layout.html
    -- index.html
```

The `info.json` file contains the theme's metadata, so that the application can provide a nice switching interface if necessary. `license.txt` is optional and contains the full text of the theme's license. `static` is served directly to clients, and `templates` contains the Jinja2 template files.

Note that exactly what templates you need to create will vary between applications. Check the application's docs (or source code) to see what you need.

1.1 Writing Templates

Flask uses the Jinja2 template engine, so you should read its documentation to learn about the actual syntax of the templates.

All templates loaded from a theme will have a global function named *theme* available to look up the theme's templates. For example, if you want to extend, import, or include another template from your theme, you can use `theme(template_name)`, like this:

```
{% extends theme('layout.html') %}
{% from theme('_helpers.html') import form_field %}
```

If the template you requested doesn't exist within the theme, it will fall back to using the application's template. If you pass *false* as the second parameter, it will only return the theme's template.

```
{% include theme('header.html', false) %}
```

You can still import/include templates from the application, though. Just use the tag without calling *theme*.

```
{% from '_helpers.html' import link_to %}
{% include '_jquery.html' %}
```

You can also get the URL for the theme's media files with the *theme_static* function:

```
<link rel=stylesheet href="{{ theme_static('style.css') }}">
```

If you want to get information about the currently active theme, you can do that with the *theme_get_info* function:

```
This theme is <a href="{{ theme_get_info('website')}}">
  <b>{{ theme_get_info('name') }}</b>
</a>
```

If you want to include things which might or might not be in your theme, you can do something like this:

```
This theme is <a href="{{ theme_get_info('website')}}">
  <b>{{ theme_get_info('name') }}</b>
  {%- if theme_get_info('version') != "" %}
    <b>({{ theme_get_info('version') }})</b>
  {% endif -%}
</a>
```

By default, this will first check if that's direct theme information, then it will check if it's part of the `options` dictionary of the theme.

1.2 `info.json` Fields

`application` [required] This is the application's identifier. Exactly what identifier you need to use varies between applications.

`identifier` [required] The theme's identifier. It should be a Python identifier (starts with a letter or underscore, the rest can be letters, underscores, or numbers) and should match the name of the theme's folder.

`name` [required] A human-readable name for the theme.

`author` [required] The name of the theme's author, that is, you. It does not have to include an e-mail address, and should be displayed verbatim.

`description` A description of the theme in a few sentences. If you can write multiple languages, you can include additional fields in the form `description_lc`, where `lc` is a two-letter language code like `es` or `de`. They should contain the description, but in the indicated language.

`website` The URL of the theme's Web site. This can be a Web site specifically for this theme, Web site for a collection of themes that includes this theme, or just the author's Web site.

`license` A simple phrase indicating your theme's license, like `GPL`, `MIT/X11`, `Public Domain`, or `Creative Commons BY-SA 3.0`. You can put the full license's text in the `license.txt` file.

`license_url` If you don't want to include the full text in the `license.txt` file, you can include a URL for a Web site where the text can be viewed. This is good for long licenses like the GPL or Creative Commons licenses.

`preview` A preview image for the theme. This should be the filename for an image within the `static` directory.

`doctype` The version of HTML used by the theme. It can be `html4`, `html5`, or `xhtml`. The application can use this to do things like switch the output format of a markup generator. (The default if this is left out is `html5` to be safe. HTML5 is used by the majority of Flask users, so it's best to use it.)

`version` This is simply to make it easier to distinguish between what version of your theme people are using. It's up to the theme/layout to decide whether or not to show this, though.

`options` If this is given, it should be a dictionary (object in JSON parlance) containing application-specific options. You will need to check the application's docs to see what options it uses. (For example, an application like a pastebin or wiki that highlights source code may want the theme to specify a default Pygments style in the options.)

1.3 Tips for Theme Writers

- Always specify a doctype.
- Remember that you have to use double-quotes with strings in JSON.
- Look at the non-theme templates provided with the application. See how they interact.
- Remember that most of the time, you can alter the application's appearance completely just by changing the layout template and the style.

CHAPTER 2

Using Themes in Your Application

To set up your application to use themes, you need to use *Themes* (`flask.ext.themes2.Themes`), in one of two ways:

```
# The first way
app = Flask(__name__)
Themes(app, app_identifier="...")

# The second way
app = Flask(__name__)
t = Themes()
t.init_themes(app, app_identifier="...")
```

The first is simply a quicker way of the second, as it will automatically call *init_themes* on your app.

This does three things:

- Adds a *ThemeManager* instance to your application as *app.theme_manager*.
- Registers the `theme` and `theme_static` globals with the Jinja2 environment.
- Registers the `_themes` module or blueprint (depending on the Flask version) to your application, by default with the URL prefix `/_themes` (you can change it).

Warning: Since the "Blueprints" mechanism of Flask 0.7 causes headaches in module compatibility mode, *init_themes* will automatically register `_themes` as a blueprint and not as a module if possible. If this causes headaches with your application, then you need to either (a) upgrade to Flask 0.7 or (b) set `Flask<0.7` in your requirements.txt file.

2.1 Theme Loaders

init_themes takes a few arguments, but the one you will probably be using most is `loaders`, which is a list of theme loaders to use (in order) to find themes. The default theme loaders are:

- *packaged_themes_loader*, which looks in your application's `themes` directory for themes (you can use this to ship one or two default themes with your application)
- *theme_paths_loader*, which looks at the *THEME_PATHS* configuration setting and loads themes from each folder therein

It's easy to write your own loaders, though - a loader is just a callable that takes an application instance and returns an iterable of *Theme* instances. You can use the *load_themes_from* helper function to yield all the valid themes contained within a folder. For example, if your app uses an "instance folder" like Zine that can have a "themes" directory:

```
def instance_loader(app):
    themes_dir = os.path.join(app.instance_root, 'themes')
    if os.path.isdir(themes_dir):
        return load_themes_from(themes_dir)
    else:
        return ()
```

2.2 Rendering Templates

Once you have the themes set up, you can call in to the theme machinery with *render_theme_template*. It works like *render_template*, but takes a `theme` parameter before the template name. Also, *static_file_url* will generate a URL to the given static file.

When you call *render_theme_template*, it sets the "active template" to the given theme, even if you have to fall back to rendering the application's template. That way, if you have a template like `by_year.html` that isn't defined by the current theme, you can still

- extend (`{% extends theme('layout.html') %}`)
- include (`{% include theme('archive_header.html') %}`)
- import (`{% from theme('_helpers.html') import show_post %}`)

templates defined by the theme. This way, the theme author doesn't have to implement every possible template - they can define templates like the layout, and showing posts, and things like that, and the application-provided templates can use those building blocks to form the more complicated pages.

2.3 Selecting Themes

How exactly you select the theme will vary between applications, so Flask-Themes2 doesn't make the decision for you. If your app is any larger than a few views, though, you will probably want to provide a helper function that selects the theme based on whatever (settings, logged-in user, page) and renders the template. For example:

```
def get_current_theme():
    if g.user is not None:
        ident = g.user.theme
    else:
        ident = current_app.config.get('DEFAULT_THEME', 'plain')
    return get_theme(ident)

def render(template, **context):
    return render_theme_template(get_current_theme(), template, **context)
```

Warning: Make sure that you *only* get *Theme* instances from the theme manager. If you need to create a *Theme* instance manually outside of a theme loader, that's a sign that you're doing it wrong. Instead, write a loader that can load that theme and pass it to *init_themes*, because if the theme is not loaded by the manager, then its templates and static files won't be available, which will usually lead to your application breaking.

2.4 Tips for Application Programmers

- Provide default templates, preferably for everything. Use simple, unstyled HTML.

- If you find yourself repeating design elements, put them in a macro in a separate template. That way, theme authors can override them more easily.
- Put class names or IDs on any elements that the theme author may want to style. (And by that I mean all of them.) That way they won't have to override the template unnecessarily if all they want to do is right-align the meta information.

CHAPTER 3

API Documentation

This API documentation is automatically generated from the source code.

class `flask.ext.themes2.`**`Themes`**(*app=None*, ***kwargs*)

This is the main class you will use to interact with Flask-Themes2 on your app.

It really only implements the bare minimum, the rest is passed through to other methods and classes.

`__init__`(*app=None*, ***kwargs*)

If given an app, this will simply call init_themes, and pass through all kwargs to init_themes, making it super easy.

Parameters

- **app** – the *~flask.Flask* instance to setup themes for.
- ****kwargs** – keyword args to pass through to init_themes

`init_themes`(*app*, *loaders=None*, *app_identifier=None*, *manager_cls=None*, *theme_url_prefix='/_themes'*)

This sets up the theme infrastructure by adding a *ThemeManager* to the given app and registering the module/blueprint containing the views and templates needed.

Parameters

- **app** – The *~flask.Flask* instance to set up themes for.
- **loaders** – An iterable of loaders to use. It defaults to *packaged_themes_loader* and *theme_paths_loader*.
- **app_identifier** – The application identifier to use. If not given, it defaults to the app's import name.
- **manager_cls** – If you need a custom manager class, you can pass it in here.
- **theme_url_prefix** – The prefix to use for the URLs on the themes module. (Defaults to `/_themes`.)

class `flask.ext.themes2.`**`Theme`**(*path*)

This contains a theme's metadata.

Parameters **path** – The path to the theme directory.

`application` = **None**

The application identifier given in the theme's info.json. Your application will probably want to validate it.

author = None
The author's name, as given in info.json. This may or may not include their email, so it's best just to display it as-is.

description = None
The human readable description. This is the default (English) version.

doctype = None
The theme's doctype. This can be `html4`, `html5`, or `xhtml` with html5 being the default if not specified.

identifier = None
The theme's identifier. This is an actual Python identifier, and in most situations should match the name of the directory the theme is in.

jinja_loader
This is a Jinja2 template loader that loads templates from the theme's `templates` directory.

license = None
A short phrase describing the license, like "GPL", "BSD", "Public Domain", or "Creative Commons BY-SA 3.0".

license_text
The contents of the theme's license.txt file, if it exists. This is used to display the full license text if necessary. (It is *None* if there was not a license.txt.)

license_url = None
A URL pointing to the license text online.

localized_desc = None
This is a dictionary of localized versions of the description. The language codes are all lowercase, and the `en` key is preloaded with the base description.

name = None
The theme's name, as given in info.json. This is the human readable name.

options = None
Any additional options. These are entirely application-specific, and may determine other aspects of the application's behavior.

path = None
The theme's root path. All the files in the theme are under this path.

preview = None
The theme's preview image, within the static folder.

static_path
The absolute path to the theme's static files directory.

templates_path
The absolute path to the theme's templates directory.

version = None
The theme's version string.

website = None
The URL to the theme's or author's Web site.

`flask.ext.themes2.`**render_theme_template**(*theme*, *template_name*, *_fallback=True*, ***context*)
This renders a template from the given theme. For example:

```
return render_theme_template(g.user.theme, 'index.html', posts=posts)
```

If *_fallback* is True and the template does not exist within the theme, it will fall back on trying to render the template using the application's normal templates. (The "active theme" will still be set, though, so you can try to extend or include other templates from the theme.)

Parameters

- **theme** – Either the identifier of the theme to use, or an actual *Theme* instance.
- **template_name** – The name of the template to render.
- **_fallback** – Whether to fall back to the default

`flask.ext.themes2.`**`static_file_url`**(*theme*, *filename*, *external=False*)

This is a shortcut for getting the URL of a static file in a theme.

Parameters

- **theme** – A *Theme* instance or identifier.
- **filename** – The name of the file.
- **external** – Whether the link should be external or not. Defaults to *False*.

`flask.ext.themes2.`**`get_theme`**(*ident*)

This gets the theme with the given identifier from the current app's theme manager.

Parameters **ident** – The theme identifier.

`flask.ext.themes2.`**`get_themes_list`**()

This returns a list of all the themes in the current app's theme manager, sorted by identifier.

3.1 Loading Themes

class `flask.ext.themes2.`**`ThemeManager`**(*app*, *app_identifier*, *loaders=None*)

This is responsible for loading and storing all the themes for an application. Calling *refresh* will cause it to invoke all of the theme loaders.

A theme loader is simply a callable that takes an app and returns an iterable of *Theme* instances. You can implement your own loaders if your app has another way to load themes.

Parameters

- **app** – The app to bind to. (Each instance is only usable for one app.)
- **app_identifier** – The value that the info.json's *application* key is required to have. If you require a more complex check, you can subclass and override the *valid_app_id* method.
- **loaders** – An iterable of loaders to use. The defaults are *packaged_themes_loader* and *theme_paths_loader*, in that order.

`bind_app`(*app*)

If an app wasn't bound when the manager was created, this will bind it. The app must be bound for the loaders to work.

Parameters **app** – A *~flask.Flask* instance.

`list_themes`()

This yields all the *Theme* objects, in sorted order.

`loaders` = **None**

This is a list of the loaders that will be used to load the themes.

`refresh``()`

This loads all of the themes into the *themes* dictionary. The loaders are invoked in the order they are given, so later themes will override earlier ones. Any invalid themes found (for example, if the application identifier is incorrect) will be skipped.

`themes`

This is a dictionary of all the themes that have been loaded. The keys are the identifiers and the values are *Theme* objects.

`valid_app_id``(`*app_identifier*`)`

This checks whether the application identifier given will work with this application. The default implementation checks whether the given identifier matches the one given at initialization.

Parameters **app_identifier** – The application identifier to check.

`flask.ext.themes2.`**`packaged_themes_loader`**`(`*app*`)`

This theme will find themes that are shipped with the application. It will look in the application's root path for a `themes` directory - for example, the `someapp` package can ship themes in the directory `someapp/themes/`.

`flask.ext.themes2.`**`theme_paths_loader`**`(`*app*`)`

This checks the app's *THEME_PATHS* configuration variable to find directories that contain themes. The theme's identifier must match the name of its directory.

`flask.ext.themes2.`**`load_themes_from`**`(`*path*`)`

This is used by the default loaders. You give it a path, and it will find valid themes and yield them one by one.

Parameters **path** – The path to search for themes in.