Templates Documentation

Release 0.0.0

Carel van Dam

Jan 12, 2021

Contents:

CHAPTER 1

Setup

1.1 Installation

Users will typically install the package directly from the Python Package Index (PyPI) while developers and contributers may prefer a source based installation.

1.1.1 Python Package Index (PyPI)

Web-Templates is hosted upon *PyPI* and is installed most conveniently through **pip**, the command line installation utility

```
pip install web-templates
```

Users looking to use the bleeding edge version of the package can install it directly from the repository

```
pip install git+https://gitlab.com/manaikan-open-source/web-templates/
```

1.1.2 Repository

The project may be cloned from the repository repository using *git*, the command line version control management utility,

```
git clone https://gitlab.com/manaikan-open-source/web-templates/ web-templates
```

and subsequently installed through *pip*

```
pip install -e .
```

1.2 Uninstallation

Similarly, the package is uninstalled using **pip**

```
pip uninstall web-templates
```

CHAPTER 2

Django

2.1 Sphinx

The *Sphinx-Django* section describes how to convert ones Sphinx sources into an intermediate target that is embeddable within Django. This section describes how to embed this intermediate source into ones pages served by Django.

2.1.1 Configuration

Given a standard Django project structure :

Website	Web Application

PROJECT
+- WEBAPP
| +- settings.py
| +- views.py
| +- urls.py
| +- ...
+- templates
  +- DOCUMENTATION
	PROJECT
	+- WEBAPP
	  +- templates
	  +- DOCUMENTATION
	  +- settings.py
	  +- views.py
	  +- urls.py
	  +- ...

Where `WEBAPP` is either the website or a web application created by `django-admin`. Perform the following operations from within the `PROJECT` root.

Builds ones documentation, setting `TARGET` to `templates/DOCUMENTATION` for a website, :

```
sphinx-build -D html_theme=sphinx_django [OPTIONS] SOURCE templates/DOCUMENTATION␣
↪[FILES...]
```

or `WEBAPP/templates/DOCUMENTATION` for a web application

```
sphinx-build -D html_theme=sphinx_django [OPTIONS] SOURCE WEBAPP/templates/
↪DOCUMENTATION [FILES...]
```

Include the `templates` folder as a source for templates for the website :

or include the web application within ones site configuration

Create a *base template* named `base.sphinx.html` (`Click to download`). Provide a view function

and register it under the url path `/documents/`.

If your setup is for a web application be sure to link across to it from the website.

Multiple sets of documentaion can be hosted provided one replaces WEBAPP and DOCUMENATION uniquely for each such application.

2.2 Configuration

Django requires that one register Web-Templates before it will acknowledge the templates that it provides. Within ones `WEBSITE/settings.py` file add `web_templates.django` under `INSTALLED_APPS` as shown

```
INSTALLED_APPS = [...
  'web_templates.django',
  ...]
```

This makes the `skeleton.html` available for extention by the *base template*, `base.html`, of ones project.

2.3 Base Template(s)

Once Web-Templates is *configured* one should create a project wide `base.html` file for their website or web application. Create the `WEBSITE/templates/base.html` and extend the `skeleton.html` as necessary.

Ideally the base template of ones web application(s), `APPLICATION/base.html`, should both require and extend this template, `base.html`, to effect their respective views.

2.4 Generic Template

Even without a standardized template for Django ones finds that some combinations of the Django packages have implemented fairly similar conventions. Certainly it seems feasible to provide a generic template for ones project that will accommodate these conventions with the skeleton provided by Web-Templates.

The following describes such a generic *base* file (`Click to download`). First it extends the **skeleton** provided by Web-Templates.

```
{% extends 'skeleton.html' %}
```

The following template processors and tags may then be enabled as one requires.

Debug It is common, under development, to view various debugging information. Django provides the debug template tag for this very purpose. Stefano suggests making this output available in the source of a page but hiding it normally.

```
{# block main_debug %}<div style="display:none;"><pre>{% debug|escape %}</pre></
↪div>{% endblock main_debug #}
```

Static Files

The *base* template makes the media and static files available globally within a projects' templates. It might be best to handle this on the application level though

```
{# load staticfiles #}
```

Site Trees Commonly one also needs to include some form of navigational component; sitetrees provides a thorough yet highly configurable structure for this.

```
{# load sitetree #}
```

CHAPTER 3

Flask

Web templates can be used with a flask application as follows

```
from flask import Flask
from webplates.flask.templates import register as register_webplates

app = Flask()
app = register_webplates(app)
```

Alternatively one may simply copy the `skeleton.html` file into ones projects' template directory.

3.1 Mixin

Web-Templates also provides s mixin class for rendering templates

```
from webplates.flask.templates import TemplateMixin

class VIEW(TemplateMixin, ...):
    ...
```

CHAPTER 4

Quart

Quart is not supported at this time. Should you wish to contribute a template please see the *Contribution* section.

CHAPTER 5

Sphinx

5.1 Django

Web-Templates provides `sphinx_django` as the compliment to Sphinxs' basic theme. To build ones documentation such that it may be embedded into Django it is most convenient to override the `html_theme` setting at compilation time.

```
sphinx-build -D html_theme=sphinx_django [OPTIONS] SOURCE TARGET [FILES...]
```

Then follow the instructions under *Django-Sphinx*.

5.1.1 Templates

The following have been observed within the Sphinx templates :

- Semantic tags are not used within their basic template for some reason; instead they rely upon `div.document` and related classes.
- `sidebar1` is structurally alongside `div.document` div while `sidebar2` is nested within it.
- Semantic tags are not used within their basic template for some reason; instead they rely upon `div.document` and related classes.

5.1.2 Skeleton

This is a complete replacement for the Sphinx Basic theme and relies heavily upon the work around described within templates:Init and Term TAGS.

The relbar element and the footer div setup by sphinx are both bundled together for the moment and clobber the `footer` tag from the layout. Ideally the footer div in the *footer* block of Sphinx should substitute the `footer_content` in the Django skeleton. Similarly `relbar2` would best be placed into the `aside` element.

5.1.3 Basic

This tries to include the Sphinx templates and includes this as a dependency. This is largely done as an example so that others might extend it for the other Sphinx themes.

Sphinx: Web Support provides the machinery to serves ones documentation through their web framework. implementation

Note: Sphinx: Web Support should be preferred over the information that follows.

Sphinx provides three means of incorporating ones documentation into ones web framework. A fourth method is considered whereby the Jinja2, the HTML builder provided by sphinx, is allowed to execute normally yet the output is decorated such that it may be consumed by a subsequent template engine.

Source processed by Sphinx	Source processed by second engine	Final output
{% block Jinja2 %} {% block ENGINE %} <html> ... </html> {% endblock ENGING %} {% endblock Jinja2 %}	{% block ENGINE %} <html> ... </html> {% endblock ENGING %}	<html> ... </html>

The sphinx templates provided by Web-Templates support this embedding of Sphinx output into some other template engine. This is down by nesting the syntax of the second engine as comments within the syntax of the first. This is an experimental feature and may or may not work in all scenarios.

5.2 Blocks

Sphinx documents the blocks that it uses within the `layout.html` file provided by its base template.

CHAPTER 6

Tornado

Tornado is not supported at this time. Should you wish to contribute a template please see the *Contribution* section.

CHAPTER 7

Template(s)

7.1 <Head>

Todo: Write

This section needs to be written :

- CSS Blocks
- Meta Blocks
- JS Blocks

7.1.1 Resets and Shivs

Due to the different Javascript and HTML implementations in different web browsers there has been a need to provide HTML "shivs", which extend the base set of tags supported by a browser, and CSS "resets" to normalize the default spacing, padding and margins of elements within a page. Presently Web-Templates do not apply any particular "shiv" or "reset" nor does it provision any block(s) for this. Should such blocks be required please open an issue.

Note: If there were a standard "shiv" or "reset" this package could certainly consider incorporating it but there seems to be no clear consensus on the matter.

7.2 <Body>

Todo: Write

This section needs to be written :

- Main structural blocks
- Main structural elements
- Review Js blocks to be more inline with the tempalates.rst documentation.

The structure described in the introduction is a bit "snoep" structurally. The <HEAD> and <BODY> elements are fleshed out by Web-Templates to be more practical. Both elements are enclosed within their own blocks and provided a minimal substructure[2] that is readily overridden.

Table 1: Actual **skeleton** and resulting *scaffold*.

Skeleton	Scaffold

`{% block html dtd %}`
`<!DOCTYPE html>`
`{% endblock dtd %}`
`{% block html %}`
`<html>`
`  {% block head %}`
`  <head>`
`  </head>`
`  {% endblock head %}`
`  {% block body %}`
`  <body>`
`  {% block  %}`
`  <header>`
`  </header>`
`  {% endblock  %}`
`  {% block  %}`
`  <nav>`
`  </nav>`
`  {% endblock  %}`
`  {% block  %}`
`  <main>`
`  </main>`
`  {% endblock  %}`
`  {% block aside %}`
`  {% endblock  %}`
`  {% block  %}`
`  <footer>`
`  </footer>`
`  {% endblock  %}`
`  </body>`
`  {% endblock body %}`
`</html>`
`{% endblock html %}`
	`<!DOCTYPE html>`
	`<html>`
	`  <head>`
	`  </head>`
	`  <body>`
	`  <header>`
	`  </header>`
	`  <nav>`
	`  </nav>`
	`  <main>`
	`  </main>`
	`  <!-- aside -->`
	`  <footer>`
	`  </footer>`
	`  </body>`
	`</html>`

The *skeleton* provides a "formally" correct HTML 5 document while allowing the user to modify this per their needs. The *<Head>* and *<Body>* sections elaborate more upon the basic structure outlined above.

[2] An `aside` block is provided so that the user may readily populate this should their layout call for it but the **skeleton** does not otherwise populate the `aside` block.

7.3 Blocks

Web-Templates tries to provide the only the necessary blocks for authors to hook into. Peppering the **skeleton** with blocks is fairly harmless since they collapse to white space in the *scaffold* once the template is processed.

7.3.1 Naming

The following naming convention is used within the **skeleton** template.

```
block  ::=   [root + "_"] + tag + ["_" + stem]
```

Within the templates the terms set aside for the template engine are kept as short and canonical as possible. Roughly speaking the terms are determined as follows :

- Use the last most unique element wherever possible e.g. the `<MAIN>` element in `<HTML><BODY><MAIN>...` is referenced as `main` in the template language references rather then `html_body_main`.
- **Repeated blocks are ideally distinguished by their enclosing tag where possible e.g. `head_js` and `body_js` is preferrabl**
  - Default/fallback blocks may also be identified e.g. `js`
- Repeated blocks are otherwise distinguished by their intent e.g. the `css_site`, `css_page` and `css_view` blocks separately identify the CSS code for the site itself, the current page and its different representations of that page e.g. monitor, mobile or print and horizontal or vertical layout.
- Where is no canonical location for a block and alternate such locations exist then it is preferable to omit the default location e.g. use `head_js_...` and `foot_js_...` instead of `js_....`

Root(s)

The `weplates:root` that precedes the `weplates:tag` is used to distinguish a group of tags.

```
root  ::=   "head" | "body" | "foot"
```

This splits the **skeleton** into three parts, "head" and "body " mapping respectively to the `<head/>` and `<body/>` tags in HTML. "foot" is included by convention but has no analogue in HTML.

Javascript

`foot` proves quite useful in placing javascript exclusively after the body, `foot_js_...` as opposed to within the head, `head_js_....` `head_js_...` is synonymous with the more canonical `js_...` blocks since users should really place their scripts in the `<head/>` setting the `deferred` attribute to delay their execution when required.

Example

Javascript libraries, commonly, are included in either the `<head>` or at the end of the `<body>` element within a document. Correspondingly Web-Templates provides both `head_js` and `body_js` but it does not provide a default `js` block since it should fall to the author to elect one location over the other. Authors might consdier overriding one location with a default block substitution e.g. One authors `{% block head_js %}{% block js %}{% endblock js %}{% endblock head_js %}` would clash with anothers `{% block body_js`

`%}{% block js %}{% endblock js %}{% endblock body_js %}` but this should be discouraged.

Note: Were a default `js` block to be included the canonical location appears to be within `<head>` element. Importing the code at the end of the `<body>` element appears to be a browser/SEO optimization for the most part; specifically it makes the page structure avilable prior to loading any code that relies upon it.

Tag(s)

`Tags` are used to identify a group of related blocks within the **skeleton**.

```
tag  ::=  html | meta | framework
```

These are split into roughly three groups.

HTML Tag(s)

These are named after the equivalent HTML tag that they enclose possible [#tags].

```
html  ::=  "header" | "nav"" | "main" | "aside" | "footer"
```

Meta Tag(s)

This is largely for meta information that is passed between the server and the client.

```
meta  ::=  "server" | "client" | "meta" | "css" | "rss" | "js"
```

The exceptions are for more esoteric blocks

browser (Rename to client) Browser specific settings

server (Not implemented) Server specific settings

meta Provides various meta information .e.g. editorial, locale, location, caching, refresh etc.

scripts The meta section also handles the various scripts e.g. Cascading style sheets, Javascript, RSS Feeds.

Framework Tag(s)

The framework tags prevent parts of the framework from clobbering other parts of the framework by splitting up where the various parties include their modification for each template.

```
framework  ::=  "site" | "page" | "view"
```

This splits the tags up as follows

site Site wide settings that are applicable to all pages. Assigned by the website author in their *base* template; extending

the **skeleton**.

page Extentions specifci settings that are applicable to a group of related pages Assigned by the extention author in their *base* template; this extends the *base* template of the website.

view Settings that are specific to a particular view. This serves as a hook for the website author to make any last adjustments for a given extention.

Stem(s)

The stem allows for specialization of the block name.

```
stem  ::=   part | item | end
```

There are three such specializations.

Part(s)

The `weplates:part` succeeds the `weplates:tag` to distinguishes the individual elements of a group of tags.

```
part  ::=   "head" | "body" | "foot"
            "prefix" | "content" | "suffix"
```

`Parts` are used universally to split a tag into three parts, namely the `head`, `body` and `foot` (Or synonmously the `prefix`, `contenst` and `suffix`). This allows users control over which part, within a tag group, their code belongs.

Item(s)

Items are highly specific and do not generalize especially well. They are mostly used to identify a specific object for substitution.

```
item  ::=   ITEM
```

End(s)

A curious scenario arises with certain template engines; when one template inherits another substituting an enclosing block it becomes necessary for child template to fully re-implement the block in the parent template.

Example

Consider modification of the opening tag in an `html` block. The child template has to re-implement the entire structure of the parent. In the example below the `head` block is accidentally excluded.

Parent	Child

{% block html %}
<html OLD >
  {% block head %}<head></head>{%
↪endblock head %}
  {% block body %}<body></body>{%
↪endblock body %}
</html>
{% endblock html %}
	{% block html %}
	<html NEW >
	  {% block body %}<body></body>{%
	↪endblock body %}
	</html>
	{% endblock html %}

Jinja2 provides a `super()` method that pulls in the structure of the parent block one is inheriting/extending. The Django Template Engine by contrast provides no such mechanism.

Web-Templates resolves this with a workaround that wraps all of the opening and closing tags with their own blocks, hence the `webplates:end` terms in block names.

```
end  ::=  "init" | "term"
```

By consistently wrapping its tags the parent effectively "unwraps" the inner blocks.

Example

Application to the example above eliminates the error within the child template.

Parent	Child

{% block html %}
{% block html_init %}<html>{% endblock
↪html_init %}
  {% block head %}<head></head>{%
↪endblock head %}
  {% block body %}<body></body>{%
↪endblock body %}
{% block html_term %}</html>{% endblock
↪html_term %}
{% endblock html %}
	{% block html_init %}<html>{% endblock
	↪html_init %}
	  {% block body %}<body></body>{%
	↪endblock body %}
	{% block html_term %}</html>{% endblock
	↪html_term %}

Default(s)

In some cases a loose convention is presumed e.g. `content` or `main_content` if this is observed to be fairly prevalent and consistent then Web-Templates will nest the block within the canonical variant within the templates. Such practice is done for convenience but heavily discouraged.

footnotes

CHAPTER 8

Extentions

Todo: Review

This section is dated and needs to be reviewed and rewritten.

3rd parties providing additional data, fitting within the scope of this package, are welcome to do so provided they do not overwrite the provided templates. Authors seeking to override these templates may do so by raising an issue accordingly upon the gitlab interface for the repository. Ideally 3rd parties would name their package with reference to this one, `template-PACKAGE`, but this is not always feasible especially when the templates are provided as part of a larger unrelaetd package. In this case one asks that authors make reference to this package within their documentation, include it as a dependency within their `setup.py` script. Ideally such templates would then reside under a subpackage/folder as in templates.PACKAGE.*. Major frameworks may ontribute a template within the root namespace but should inform the maintainers of `templates` by raising an issue. Should any conflicts arise as a result these will be resolved by discussion betwen the parties involved or if no discussion is possible then the first party reserving a name will win.

CHAPTER 9

Design

Todo: Review

This section is dated and needs to be reviewed and rewritten.

9.1 Base Files

Todo: These base files are named `base.ENGINE.html`; this should be applied for all of the base frameworks. It frees the user up to migrate from some `base.html` to `base.ENGINE.html` until they can adopt web templates across their entire site and revert back to `base.html`. The "Sphinx in Django" section of the documentation also illustrates the utility in having the `sphinx_ENGINE` themes compile ones documentation to depends upon a `base.ENGINE.html` template rather then a `base.html`.

I believe these templates were measnt to 'passivate'; that is setup the base functionality for the website for a given template engine e.g. include boostrap/material design react etc.

In the package root there are a set of `base.ENGINE.html` files. i have forgotten their purpose to be honest but i believe they are meant to generate the **skeleton** files that are ready to be processed by the Django Template Engine (DTE) after the other template engine has processed the file.

```
base.ENGINE.html > FRAMEWORK/ENGINE > TEMPLATE.html(DTE) > Django Template Engine >␣
↪PAGE.html
```

Consider a document generated by Sphinx, the base.sphinx.html template, creates a PAGE.html(DTE) page that can then be processed by the Django template engine to produce the final output. This allows both template engines to process the page in its usual manner but the content flows "upstream" until the final production.

Todo: Specifically for sphinx the file `E:Pythonweb-templatesweb_templatessphinxdjangolayout.html`, a copy of `E:Pythonweb-templatesweb_templatesbasiclayout.html`, may be sued

by sphinx to compile documentation that depends upon a `base.sphinx.html` file; based upon the users `base.html` which in turn subclasses `web-teplates/../skeleton.html`. The current `E:Pythonweb-templatesweb_templatesbase.sphinx.html` seems to duplicate the `E:Pythonsphinxsphinxthemesbasiclayout.html` from the Sphinx base theme more then anything else; I believe this was to be merged with `E:Pythonweb-templatesweb_templatessphinxdjangolayout.html`. `E:Pythonweb-templatesweb_templatesbase.sphinx.html` was probably meant to serve as a bridging template for users to simply copy or extend as necessary.

9.2 Structure

The templates are organized within the project as follows

9.2.1 Naming

The **skeleton** templates provided by Web-Templates for the *base* templates of other projects to inherit are named accordingly to the following scheme

```
BACKEND_FRAMEWORK[/EXTENTION(S)][/FRONTEND_FRAMEWORK][/EXTENTION(S)]/skeleton.
↪TEMPLATE_ENGINE.html
```

9.3 Django

9.3.1 Name Spaced Packages

Django does not pick up templates within a namespaced package, like `templates`, especially when there are multiple paths as is common with 3rd parties adding additional functionality. Providing templates through the following structure

```
templates/
 base.html
```

resulted in the following message, for example

```
django.core.exceptions.ImproperlyConfigured: The app module <module 'templates'␣
↪(namespace)> has multiple filesystem locations (['..\\web-templates\\templates', '..
↪\\other\\templates']); you must configure this app with an AppConfig subclass with␣
↪a 'path' class attribute.
```

This implies one might select a preferred package by specifying a preferred root package within `AppConfig`.

A work around is to nest the template one level deeper, removing the ambiguity and appeasing Django accordingly. Consequently web-tamples has been refactored from the original structure to the following one

```
templates/
 django/
  base.html
```

resolving the issue. Under the hood Python is selecting the unique subpackage, `django`, from the multi-targeted namespace package, `templates`, that contains it; eliminating the ambiguity and finding the template accordingly.

Hence the need to use `templates.django` within the middleware of a projects `settings.py` file.

CHAPTER 10

Contribution

Todo: Review

This section is dated and needs to be reviewed and rewritten.

10.1 Structure

10.1.1 Package Structure

The package structure is outlined below and serves as a guideline for those providing competing and/or complementary features.

```
Project/                          # Project/Repsoitory root
 templates/                       # Namespace reserved for web-templates
  django/                         # Namespace reserved for Django templates
    templates/skeleton.html       # Django skeleton template
  flask/                          # Namespace reserved for Flask templates
  tornado/                        # Namespace reserved for Tornado templates
  sphinx/                         # Namespace reserved for sphinx templates
  django.html                     # Reserved name for a Django template engine template
  jinja.html                      # Reserved name for a Jinja/Jinja2 template
 setup.py                         # Setup.py script that bundles everything together
 MANIFEST.in                      # Manifest file that embeds the various data sources␣
↪into the distributed package.
```

This package reserves the package namespace `templates.*` by providing a namespaced package/folder with the the name `templates` in the project root. Furthermore the template names `django.html` and `jinja.html` are reserved for the base HTML templates used by the Django and Jinja templating systems respectively.

Note: Given the package/template path `package/django/templates/skeleton.html` one includes

package.django within INSTALLED_APPLICATIONS and extends skeleton.html within their base.html template. Django knows to look for the templates under the templates subfolder.

10.1.2 Structure limitations

The desired structure for the core of the package is as follows

```
project\
 templates\
  django\templates\skeleton.html # Combines base.structure.html and base.html into a␣
↪single file, currently available as
  sphinx\templates\skeleton.html # Combines `base.sphinx.html`, `basic.layout.html`␣
↪and `basic.layout - copy.html` into a single file
  jinja\templates\skeleton.html  # Combines `base.jinja.html` into a single file
```

Previously the following alternatives have been attempted but they fail for various reasons.

templates\ base.html __init__.py	templates\ templates\base.html __init__.py
Unpackageable	Makes template into an explicit package

Django expects that a templates folder exists containing the templates. In the first structure Django simply ignores the base.html as a result. In the second case Django finds the file as appropriate but it does not like the fact that the root package is namespaced. The work around is to nest the package one level deeper to appease Django and get the user to list templates.django under their *INSTALLED_APPS* in their settings.py file.

10.1.3 Artefacts

The following templates are the leftovers from prior development attempts.

```
basic/              Additional Sphinx related templates (Not sure on status or usage)
 layout.html        This embeds the jinja2 tags expected by sphinx within the django␣
↪template tags (not sure how this differs from base.sphinx.html).
base.django.html    The django base template as prescribed by modern HTML5;␣
↪Originally base.structure.html
base.sphinx.html    This is adapted from base.structure.html to accomodate Sphinx␣
↪projects.
base.jinja.html     This is adapted from base.structure to accomodate jinja as the␣
↪template engine.
base.html           This is adapted from base.structure to accomodate JavaScript␣
↪libraries, this needs more work i.e. selectively enabling one or another library.
```

For the most part these are to be deprecated/deleted but there is still some value to extracted before doing so. The templates under the basic should really be merged/compared against the base.sphinx,html file for instance as this has allowed me to embed sphinx documentation within a Django site before for instance.

10.2 HTML

The *W3C* defines the specification for HTML, CSS and Javascript (a.k.a. ECMA script). This section tries to summarize/link to the relevant information that they provide.

10.2.1 W3C

The *W3C* has setup the Web accessiblity Initiative which aims to make the web more accessible to everyone. The project should follow these recommendations as best as possible.

10.2.2 Schema

The Schema or Data Vocabulary provides assistive information to the search engine.

10.2.3 DTD

The W3 Schools lists the HTML tags supported by each DTD type.

10.3 Packages

10.3.1 Name Spaced Packages

Classic/Conventional Python packages consist of a folder containing a, typically empty, `__init__.py` file to identify them as such. Python 3.4 introduced namespaced packages which consist of a folder containing further sub-packages or `MODULE.py` file(s), none of which are named `__init__.py`. Conveniently `setuptools` has allowed package writers to embed "data" files within their packages for sometime. Combining these properties together affords one the ability to distribute Python packages that, ironically, contain no Python code.

Furthermore namespaced packages act as mixins. That us any two modules nested under a common package path, neither of which may contain an `__init__.py` file, may be imported simultaneously as though they were part of the same package.

Example :

As an example consider the two files, `MODULE_A.py` and `MODULE_A.py`; both of which are nested under a folder with the same name, `PACKAGE`, but located at different places upon ones file system.

```
this/project      # Some path in sys.paths
 PACKAGE/         # Common package path
  ...             # Other modules none of which are named __init__.py
  MODULE_A.py     # Module in the namespace'd set
that/project      # Some path in sys.paths
 PACKAGE/         # Common package path
  ...             # Other modules none of which are named __init__.py
  MODULE_B.py     # Module in the namespace'd set
```

Python views both files as though they were modules, `MODULE_A` and `MODULE_B`, under the same namespaced package, `PACKAGE`, making it possible to perform the following imports

```
from PACKAGE import MODULE_A
from PACKAGE import MODULE_B
```

This is not possible with classical Python packages, in fact adding an `__init__.py` file to either package breaks the above import statements.

Effectively one may create a namespaced package that contains only *data* and will merge well with other packages that do the same. Under the mild restriction that such packages do not include an `__init__.py` file within their structure.

10.4 Tools

This section reviews some of the tools available to the web developer.

10.4.1 HTML Validation

HTML validation can be done through the W3.org validator .

Symbolic Links

Symbolic links may be made on most systems these days. To do so on windows use the following

```
mklink /J TARGET SOURCE
```

While linux users should employ

```
ln -s source target
```

Note: Originally the templates where made available to the projects one was sorking on by symbolic links. This made the same files available to multiple projects with different requirements and proved to be very inflexible and prompted the proper packaging of these templates.

10.4.2 Crimson Editor

The Jinja template is generated from the Django template by copying it and performing the following substitutions.

```
    {%([^%]*)%}      ->       {{'{%\1%}'}}
regular expression       replacement pattern
```

Note: The regular expression and assosciated pattern are specific to crimson editor.

10.5 Templates

Django used to search the `PROJECT/WEBSITE/templates` folder for common templates, it no longer does so and expects a `templates` folder to reside in the project root, `PROJECT/templates`. Should one opt to their templates within the website directory, under `PROJECT/WEBSITE/templates`, then they should include them within their project settings by assigning the `templates[dirs]` attribute in the `WEBSITE/settings.py` file.

```
TEMPLATES = {...
 DIRS = ['website/templates'],
...}
```

Name Spaced Packages discusses how one might distribute common template/data through Python packages. Such packages, and related 3rd party extentions, allow for unhindered contribution, provided all parties adhere to a few conventions. *Package Structure* outlines the conventions for the web-templates package so that no one overrides or breaks anothers contribution.

Todo: Review

This section is dated and needs to be reviewed and rewritten.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/web-templates/checkouts/latest/docs/contribution.rst, line 5.)

Todo: Review

This section is dated and needs to be reviewed and rewritten.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/web-templates/checkouts/latest/docs/design.rst, line 5.)

Todo: These base files are named `base.ENGINE.html`; this should be applied for all of the base frameworks. It frees the user up to migrate from some `base.html` to `base.ENGINE.html` until they can adopt web templates across their entire site and revert back to `base.html`. The "Sphinx in Django" section of the documentation also illustrates the utility in having the `sphinx_ENGINE` themes compile ones documentation to depends upon a `base.ENGINE.html` template rather then a `base.html`.

I believe these templates were measnt to 'passivate'; that is setup the base functionality for the website for a given template engine e.g. include boostrap/material design react etc.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/web-templates/checkouts/latest/docs/design.rst, line 12.)

Todo: Specifically for sphinx the file `E:Pythonweb-templatesweb_templatessphinxdjangolayout.html`, a copy of `E:Pythonweb-templatesweb_templatesbasiclayout.html`, may be sued by sphinx to compile documentation that depends upon a `base.sphinx.html` file; based upon the users `base.html` which in turn subclasses `web-teplates/../skeleton.html`. The current `E:Pythonweb-templatesweb_templatesbase.sphinx.html` seems to duplicate the `E:Pythonsphinxsphinxthemesbasiclayout.html` from the Sphinx base theme more then anything else; I believe this was to be merged with `E:Pythonweb-templatesweb_templatessphinxdjangolayout.html`. `E:Pythonweb-templatesweb_templatesbase.sphinx.html` was probably meant to serve as a bridging template for users to simply copy or extend as necessary.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/web-templates/checkouts/latest/docs/design.rst, line 29.)

Todo: Review

This section is dated and needs to be reviewed and rewritten.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/web-templates/checkouts/latest/docs/extentions.rst, line 5.)

Todo: Review how to show a Software Application, Bread Crumb, FAQ and How T links.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/web-templates/checkouts/latest/docs/seo.rst, line 15.)

Todo: Write

This section needs to be written :

- Main structural blocks
- Main structural elements
- Review Js blocks to be more inline with the tempalates.rst documentation.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/web-templates/checkouts/latest/docs/templates/body.rst, line 7.)

Todo: Write

This section needs to be written :

- CSS Blocks
- Meta Blocks
- JS Blocks

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/web-templates/checkouts/latest/docs/templates/head.rst, line 7.)

CHAPTER 11

Glossary

CSS (Cascading Style Sheets) These code embedded within a webpage that specifies how it ought to look

Django A web framework written in Python

DTD (Document Type Declaration) A tag at the top of an HTML document declaring it as one or other variant of HTML

DTE (Django Template Engine) The engine that processes the *DTL*

DTL (Django Template Language) The Django Template Lanugage

Flask A micro-web framework written in Python

HTML (Hyper Text Markup Language) This is the underlying format for a webpage

Jinga Jinga and Jinga II are standalone template engines that a number of Python frmeworks have leveraged instead of writing their own.

Template Engine Django provides it's own template engine that, to ones knowledge, is known simply as the template engine. This engine is used only by django and has not been ported to other frameworks

Tornado A web framework originating from the twisted project

Python Seriously ?!? A programing language focused upon code readability rather then wearing out the punctuation keys upon ones key board

PyPI (Python Package Index) The Python package index, a.k.a. The Cheese Shop after the Monty Python Sketch, hosts all of the publicly available Python packages.

Sphinx Pythons' documentation generation utility, often useful in the documentation of other packages in other languages.

W3C (World Wide Web Consortium) The world wide web consortium are a body of ICANN, essentially the people that define the internet

WAI (Web Accessiblity Initiative) The web accessibility initiative aims to make the web more acccessible to persons with dissabilities

Problem

There isn't a standardized *base* template that the authors(s) of websites and the author(s) of extentions for a given web framework can collectively reference. The website author specifies the *base* template for their website per their requirements, populating it with the components provided by framework extentions. The authors of framework extentions can not know the final structure of the *base* template for a given website and must assume which block such templates might provide.

Objective

The package is to provide a standard structure/methodology for both the website and extention author(s) of the respective frameworks.

Scope

Web-Templates focuses primarily upon the Python web frameworks relying upon templating mechanism

Proposal

A **skeleton** template that both the authors of websites and of framework extentions can reference for their chosen framework. This has the following benefits :

- The webiste author(s) using these frameworks, and their extention packages, can customize their *base* template, that extends the **skeleton** template, against a standardized set of guidelines.
- The extention author(s) of these frameworks can, reasonably, presume the existance of a standardized *base* template.

Solution

Web-Templates provides a **skeleton** with the minimal structure necessary to produce a standardized *scaffold* for the HTML documents for use with Python's respective Web Frameworks (Django/Tornado/Flask/etc.). The "ideal" **skeleton** should produce the following *scaffold* after being passed through ones template engine[1] :

[1] The package actually extends this "ideal" **skeleton** to provide a template with a more practical alternative.

Table 1: Ideal **skeleton** and resulting *scaffold*.

Skeleton	Scaffold
{% block html dtd %} <!DOCTYPE html> {% endblock dtd %} {% block html %} <html> {% block head %} <head> </head> {% endblock head %} {% block body %} <body> </body> {% endblock body %} </html> {% endblock html %}	<!DOCTYPE html> <html> <head></head> <body></body> </html>

Organization

The documentation is partitioned into sections that are pertinent to the package users or website authors, third parties providing framework extentions and package contributors.

Setup describes the *installation* and *removal* of Web-Templates. Then following section(s), each dedicated respectively to a framework, details the configuration and usage of Web-Templates within ones project(s). Each section prescribes a standardized *base* template, where possible, and details the underlying **skeleton** that it extends.

Third parties incorporating Web-Templates into their projects should read the *extentions* section along with the section relevant to the framework(s) they support. The rational, nomenclature and structure of the **skeleton** is described in *Template*.

Finally the package *design* is described and how one might *constribute*.

Note: This package only supports Python 3.4 and onwards as it relies upon name-spaced packages (See **PEP 420** and *Name Spaced Packages*).

---

footnotes

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Index