django-formrenderingtools Documentation

Release 0.2.3

Benoit Bryon

April 24, 2016

Contents

Warning: This project is inactive. Django-floppyforms is the recommended alternative since it supports form layouts. See: https://pypi.python.org/pypi/django-floppyforms

Customize layout of Django forms in templates, not in Python code.

Rather than using {{ form.as_p }}, set up and reuse templates to render Django's form elements.

As an example, to reproduce Django's `{{ form.as_p }}`:

```
{% load form_layouts %}
<form>
  {% form layout="as_p" %}
  <input type="submit">
</form>
```

CHAPTER 1

Resources

- Online documentation: http://packages.python.org/django-formrenderingtools/
- PyPI page: http://pypi.python.org/pypi/django-formrenderingtools
- Source code repository: http://bitbucket.org/benoitbryon/django-formrenderingtools
- Bugtracker: http://bitbucket.org/benoitbryon/django-formrenderingtools/issues

CHAPTER 2

Contents

2.1 Overview

This Django application provides tools for the web designer to customize the rendering of forms in templates.

2.1.1 Rendering forms with Django is boring

For an introduction to Django forms, see the Django documentation about forms.

The following section will point out some limitations of the Django's standard way to display forms in templates. Here is a short list of common problems:

- how to add CSS classes to the form elements?
- how to display the fields in a different order than specified in the form's python class definition?
- how to display only a subset of the form? How to display fieldsets?
- how to customize only a few fields in the form?

If you are already aware of these problems, you can read about *the form_layouts template tag library* in the next section.

Otherwise, let's begin by considering the following Django form:

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField()
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

Django's standard way to render forms is to use the form.as_p() method or similar:

```
{{ my_form.as_p }}
```

You get something like this:

```
<p>
  <label for="id_subject">Subject:</label>
  <input id="id_subject" type="text" name="subject" maxlength="100" />
</p>
<p>
```

```
  <label for="id_message">Message:</label>
  <input type="text" name="message" id="id_message" />
</p>
<p>
  <label for="id_sender">Sender:</label>
  <input type="text" name="sender" id="id_sender" />
</p>
<p>
  <label for="id_cc_myself">Cc myself:</label>
  <input type="checkbox" name="cc_myself" id="id_cc_myself" />
</p>
```

Seems magic...

HTML output is fully controlled by Python scripts. This is efficient, but the template designer does not have control on it. This does not contribute to the separation between logic and design.

Now, what if the template designer wants to add a "required" CSS class attribute to the required fields in a form? He has to write code. As explained in the the Django documentation about form templates, you can write a loop:

```
{% for field in form %}
<div class="fieldWrapper{% if field.field.required %} required{% endif %}">
  {{ field.errors }}
  {{ field.label_tag }}: {{ field }}
</div>
{% endfor %}
```

Now what if the template designer want to customize only one field in the form? He has to write down the complete form. Here is an example where subject's help text is displayed before field, and message label is "Your message":

```
<div class="fieldWrapper">
  {{ form.subject.errors }}
  {{ form.subject.label_tag }}:
  <span class="helptext">{{ form.subject.help_text }}</p>
  {{ form.subject }}
</div>
<div class="fieldWrapper">
  {{ form.message.errors }}
  <label for="id_message">Your message:</label>
  {{ form.message }}
  <span class="helptext">{{ form.message.help_text }}</p>
</div>
<div class="fieldWrapper">
  {{ form.sender.errors }}
  {{ form.sender.label_tag }}:
  {{ form.sender }}
</div>
<div class="fieldWrapper">
  {{ form.cc_myself.errors }}
  {{ form.cc_myself.label_tag }}:
  {{ form.cc_myself }}
</div>
```

And what if you want to customize many fields in many forms? Then it gets really boring, isn't it?

The Django documentation about reusable form templates says:

```
If you find yourself doing this often, you might consider creating a custom
inclusion tag.
```

So, here comes django-formrenderingtools. This application uses templates to render forms, wherever it is possible. Default templates can be reused, and specific templates can be created if necessary.

2.1.2 Introducing the "form_layouts" template tag library

Consider the form used in the previous section.

Here is how you can use django-formrenderingtools to display it:

```
{% load form_layouts %}
{% form %}
```

Now, if you want to customize the "subject" field, create a "form_layouts/contact/fields/subject.html" template. Write "Hello world!" in this template.

By doing this, you have created a "contact" form layout (the directory name) and a custom template for the "subject" field. So the new template code is:

```
{% load form_layouts %}
{% form layout="contact" %}
```

Refresh your page. You should see "Hello world!" instead of the subject field.

Notice that the template name is important. If the template does not have the "good" name, then the template tag will not be able to find it. See Template names and directory structure for details.

Have a look on the "django_formrenderingtools/templates/form_layouts/default/" directory to get the default templates used by the "form_layouts" template tags.

You can customize more fields, forms, labels by creating templates.

You can reuse the "contact" form layout for other forms.

You can override the default form layout so that it fits your coding conventions.

Learn more by reading the Reference or Demo project sections of this documentation.

2.1.3 Concepts

The goal of this application is to provide a pack of template tags which helps you render each element in a form: full form, list of fields, non field errors (global errors), field errors (specific errors), field, label, help text... It is intended to be simple to learn and easy to extend.

Every form element has a corresponding template tag, which uses templates to generate the output. Template designers no longer rely on developers to customize the form output.

This application provides a "form_layouts" template tag library which itself provides the following template tags:

- form: renders a full form, i.e. non field errors, all fields, field errors, labels and help texts
- form_errors: renders global form errors, i.e. non field errors
- field_list: renders a set of fields in a form, with corresponding field errors, labels and help texts
- field: renders a field, with field errors, label and help text
- field_errors: renders errors related to a field
- label: renders a field's label
- help_text: renders a field's help text

For a deeper description of each template tag, see Template tags.

This application uses a template-naming system that lets you reuse generic templates or use specific ones, depending on your needs. You can reuse built-in templates, override them or create your own templates. See Template names and directory structure for details.

2.2 Installation

The code is published under the BSD license. See License for details.

If you just want to discover the application, you may have a look at the Demo project.

If you want to contribute to the code, you should go to Contribution guidelines documentation.

2.2.1 Install Python package

Install the package with your favorite Python installer. As an example, with pip:

```
pip install django-formrenderingtools
```

2.2.2 Update Django project settings

Add `django_formrenderingtools` to the `INSTALLED_APPS` list in your Django project settings:

```
INSTALLED_APPS = (
    # ...
    'django_formrenderingtools',
    # ...
)
```

Depending on you configuration, you may also check `TEMPLATE_LOADERS` and `TEMPLATE_DIRS` to make sure that templates distributed within django-formrenderingtools are discovered.

2.3 Demo project

The `demo/` folder holds a demo project to illustrate django-formrenderingtools usage.

2.3.1 Browse demo code online

See demo folder in project's repository [1].

2.3.2 Deploy the demo

System requirements:

[1] https://bitbucket.org/benoitbryon/django-formrenderingtools/src/tip/demo

- Python [2] version 2.6 or 2.7, available as `python` command.

  Note: You may use Virtualenv [3] to make sure the active `python` is the right one.

- `make` and `wget` to use the provided `Makefile`.

Execute:

```
hg clone http://bitbucket.org/benoitbryon/django-formrenderingtools
cd django-formrenderingtools/
make demo
```

It installs and runs the demo server on localhost, port 8000. So have a look at http://localhost:8000/

Note: If you cannot execute the Makefile, read it and adapt the few commands it contains to your needs.

Browse and use `demo/demoproject/` as a sandbox.

2.3.3 Experiment

Use the demo project as a sandbox!

Note: Demo is part of the development process and part of the documentation. It has been created to both help users discover the application and developers to test features with real-world use cases.

2.4 Reference

2.4.1 Template tags

django-formrenderingtools provides the "form_layouts" template tag library, which itself provides the following template tags:

- *form*: renders a full form, with all errors (field and non field errors), fields and labels.
- *form_errors*: renders global form errors, i.e. non field errors
- *field_list*: renders a list of fields, with field errors, fields and labels. By default, uses {% field %}.
- *field*: renders a field, with field errors and label. By default, uses {% label %}.
- *field_errors*: renders errors related to a field
- *label*: renders a field's label
- *help_text*: renders a field's help text

[2] http://python.org
[3] http://virtualenv.org

form

Renders a full form, with all errors (field and non field errors), fields and labels.

By default, uses *field_list*.

Minimal usage

```
{% load form_layouts %}
{% form %}
```

In this case:

- a context variable named "form" is required. You can use {% with %} for this purpose.
- the default layout will be used
- all fields in the form will be displayed, in the order specified in the form's python class definition.

Usage with options

```
{% load form_layouts %}
{% form form=my_form layout="my_layout" fields="a_field_name,another_field" exclude_fields="some_fie
```

Input parameters

form optional, a context variable, the form instance to be rendered. If empty, the template tag searches for a context variable named "form".

layout optional, defaults to settings.FORMRENDERINGTOOLS_DEFAULT_LAYOUT ("default" by default), a context variable or a string, the layout to be used. Through this parameter, you implicitely specify the template directory to use. See Template names and directory structure for details.

fields optional, defaults to None, a list or comma-separated (no spaces allowed) string which represents the names of fields that you want to be displayed. Only those fields will be displayed. If a field is in both "fields" and "exclude_fields", then it won't be displayed.

exclude_fields optional, defaults to None, a list or comma-separated string which represents the names of fields that you do not want to be displayed. Only other fields will be displayed. If a field is in both "fields" and "exclude_fields", then it won't be displayed.

template optional, defaults to "default.html", a string, the template name to use. See Template names and directory structure for details.

form_errors

Renders non field errors of a form.

Input parameters are the same as the *form* template tag.

field_list

Renders several fields.

Input parameters are the same as the *form* template tag.

field

Renders a field: errors, label, field and help_text.

Notice that Django-formrenderingtools is not intended to customize widgets. Have a look at django-floppyforms for this purpose.

Input parameters

field optional, a context variable, the field instance to be rendered. If empty, the template tag searches for a context variable named "field".

layout optional, defaults to settings.FORMRENDERINGTOOLS_DEFAULT_LAYOUT ("default" by default), a context variable or a string, the layout to be used. Through this parameter, you implicitely specify the template directory to use. See Template names and directory structure for details.

template optional, defaults to "default.html", a string, the template name to use. See Template names and directory structure for details.

field_errors

Renders the errors attached to a field.

Input parameters are the same as the *field* template tag.

label

Renders the label of a field.

Input parameters are the same as the *field* template tag.

help_text

Renders the help text of a field.

Input parameters are the same as the *field* template tag.

2.4.2 Settings

The django-formrenderingtools settings are prefixed with "FORMRENDERINGTOOLS_".

It is recommended not to change the following settings if you are aware of the "convention over configuration" practice.

These settings were originally created for the developer convenience and to enable tests.

FORMRENDERINGTOOLS_TEMPLATE_DIR

By default:

```
FORMRENDERINGTOOLS_TEMPLATE_DIR_TEMPLATE_DIR = 'form_layouts'
```

By default, the "form_layouts" template tag library searches for templates within the "form_layouts/" folder in template directories.

You can change this behaviour by overriding FORMRENDERINGTOOLS_TEMPLATE_DIR in your project's settings.

FORMRENDERINGTOOLS_DEFAULT_LAYOUT

By default:

```
FORMRENDERINGTOOLS_DEFAULT_LAYOUT = 'default'
```

When you call a template tag without specifying the optional "layout" argument, then it fallbacks to settings.FORMRENDERINGTOOLS_DEFAULT_LAYOUT.

FORMRENDERINGTOOLS_DEFAULT_TEMPLATE

By default:

```
FORMRENDERINGTOOLS_DEFAULT_TEMPLATE = 'default.html'
```

When you call a template tag without specifying the optional "template" argument, then it fallbacks to settings.FORMRENDERINGTOOLS_DEFAULT_TEMPLATE.

2.4.3 Template names and directory structure

As introduced in the previous chapter, the "form_layouts" template tag library lets you customize several levels of form rendering. The template tag library searches for templates in a particular directory structure.

Overview

The default structure provided by Django-formrenderingtools is:

- templates => a template directory, included in settings.TEMPLATE_DIRS
  - form_layouts => a directory for the form_layouts material
    - default => the default layout
      - field => templates for fields
      - field_errors
      - field_list
      - form
      - form_errors
      - help_text
      - label

Every element in the list above is a folder. It contains one default.html template.

If you look at the templates provided by formrenderingtools, you will notice that "templates/form_layouts" also contains the following folders:

- as_ul => a layout that reproduces {{ form.as_ul }}

- as_p => a layout that reproduces {{ form.as_p }}
- as_table => a layout that reproduces {{ form.as_table }}

These layouts exist for demonstration and migration purposes. They are based on the default layout. You can read the template code to create your own layouts.

How to use other templates?

Several parameters allow you to change the locations where template selection occurs:

- the "layout" parameter of the tags in the "form_layouts" template tag library. You can use whatever you want provided it is a valid directory name, without leading and trailing slashes:

```
{% form layout="contact_form" %}
{% form layout="user_account/register" %}
```

  The "layout" parameter affects nested elements. It means that using {% form layout="contact" %} will generate implicit {% field layout="contact" %} calls.

- the "template" parameter of the tags in the "form_layouts" template tag library. You can use whatever you want provided it is a valid file name. Do not forget the extension:

```
{% form template="contact_form.html" %}
{% form template="user/register.html" %}
```

  The "template" parameter affects only the current element. It means that using {% form template="contact.html" %} will generate implicit calls with default template, as {% field template="default.html" %}.

Priority

As an example, if you didn't change the default template and layout names, {% form layout='LAYOUT/DIR' template="TEMPLATE/NAME.html" %} will use the first existing template in the following list:

- form_layouts/LAYOUT/DIR/form/TEMPLATE/NAME.html
- form_layouts/default/form/TEMPLATE/NAME.html
- form_layouts/LAYOUT/DIR/form/default.html
- form_layouts/default/form/default.html

Similar rules are used for other elements.

Additional variables

Keep in mind that, in general use case, the "layout" and "template" parameters should be enough to get the result you want. The following parameters are documented for contributors:

- settings.FORMRENDERINGTOOLS_DEFAULT_LAYOUT: allows you to change the implicit value of the "layout" parameter. Default value is "default". Notice that you can get exactly the same result by overriding the form_layouts/default/* templates: simply make sure that the templates in your project/application have priority over the ones provided by formrenderingtools.
- settings.FORMRENDERINGTOOLS_DEFAULT_TEMPLATE: allows you to change the implicit value of the "template" parameter. Default value is "default.html". Notice that you can get exactly the same result by overriding the form_layouts/default/{ELEMENT_NAME}/default.html templates: simply make sure that the templates in your project/application have priority over the ones provided by formrenderingtools.

- settings.FORMRENDERINGTOOLS_TEMPLATE_DIR. A prefix for all templates used by the django-formrenderingtools application. It is not recommended to change it, because you should be able to perform the same thing by using one of the previously described tip.

2.4.4 Cascading style sheets

The django-formrenderingtools application uses a set of default CSS classes. So you may want to write a CSS stylesheet. Here is the list of classes that are used.

General styles

The following CSS classes are applied by the default form layout:

```
ul.errorlist{} /* non field error list container */
ul.errorlist li{} /* non field error */
.formItem{} /* container for each field (field errors, label, input and help text) */
.formItem.required{} /* fields that are required */
.formItem.hasErrors{} /* fields that have errors */
.formItem ul.errorlist{} /* field error list */
.formItem ul.errorlist li{} /* field error */
.formItem label{} /* field label */
.formItem.required label{} /*  label of required fields */
.formItem.hasErrors label{} /* label of fields that have errors */
.formItem .help{} /* container for field help text */
```

You may also declare additional styles for inputs.

Notice that hidden fields are not rendered in a “formItem” container.

Field specific styles

If the input is not hidden, then the HTML name of the field is appended as a CSS class at the field container level.

As an example, customizing the CSS class ”.formItem.email” and children will affect only the form fields named “email”. So, to customize the input you will have to customize ”.formItem.email input”.

2.5 Migrate from Django’s builtins to django-formrenderingtools

How to safely migrate an existing project from standard Django’s practices to django-formrenderingtools? Here are some guidelines.

2.5.1 Install django-formrenderingtools

See Installation for details.

2.5.2 Replace {{ form }} by {% form %} in your templates

In the templates you edit, feel free to replace any:

- `{{ form }}` by `{% form layout="as_table" %}`
- `{{ form.as_p }}` by `{% form layout="as_p" %}`

- `{{ form.as_ul }}` by `{% form layout="as_ul" %}`
- `{{ form.as_table }}` by `{% form layout="as_table" %}`

Note: django-formrenderingtools' builtin "as_*" layouts reproduce Django's behavior. Tests are written to check this fact.

2.5.3 Migrate custom layouts and includes

Search for any `{{ form.* }}` occurrence in templates. You should be able to replace these occurrence with some django-formrenderingtools features.

If you already used snippets via `{% include %}`, you should consider migrating the templates to django-formrenderingtools (see Template names and directory structure):

- it proposes a convention of directory structure;
- if new features (such as template loading optimizations) are released, you'll automatically get them.

2.6 Best practices

Here are some of the guidelines about form rendering, followed in this application.

- Where a web designer contributes to project, use templates rather than Python scripts
- Put CSS classes to container, so that both container and contained element can be styled. As an example use:

  rather than:

  because the graphic designer may want the DIV's background or the INPUT's border to be red.

2.7 Contribution guidelines

2.7.1 Get a development environment

```
# Get the source.
hg clone http://bitbucket.org/benoitbryon/django-formrenderingtools
cd django-formrenderingtools/
# Install.
make develop
# You can run the tests!
make test
```

If everything went fine, you have a everything you need in the folder. Have a look at the provided `Makefile` for details.

If a problem occurred, look at the provided `Makefile`. It is the live install-for-development documentation.

2.7.2 Guidelines

- Create issues, preferably before starting to hack, so that we can discuss as soon as possible.
- Work in branches, ideally one branch per issue.

- Write tests and run them with `make test`.
- Write documentation in `docs/`. Build it with `make documentation`.
- Update the demo project in `demo/`.

2.8 About django-formrenderingtools

This project was initiated in 2008 in order to help web designers focus on templates (pseudo-HTML) and CSS, whereas developers focus on Python code.

2.8.1 Alternatives

This document lists some projects which provide similar or complementary functionalities.

django-floppyforms

django-floppyforms [1] deals with form widgets, which are not in the scope of `django-formrenderingtools`. The two projects offer complementary functionalities.

Gsoc2011: Revised form rendering

In 2011, the Revised form rendering project [2], during Google Summer of Code, tried to:

1. merge django-floppyforms [1] in core Django;
2. add similar functionality for form layouts in core Django.

Part one was done pretty quick.

Part two raised lots of discussions. The resulting plan was quite big. And (as far as I know) didn't succeed.

Even if `django-formrenderingtools` introduced the #2 basic functionalities more than one year before, it wasn't well known (I guess it remains quite invisible as of July 2012) and wasn't considered as a candidate for implementation. It was mentioned in discussions. Some concepts influenced Gsoc proposal. But not more.

The Gsoc ended, Django 1.4 was released in 2012. As of July 2012, there is no template-based form rendering in core Django.

So, `django-formrenderingtools` keeps on being an option...

Note: One big point in discussions during Gsoc was about performances: using templates can generate many disk access and some overhead. django-floppyforms [1] has benchmarks and implemented optimizations. `django-formrenderingtools` has none currently, but contributions are welcome!

---

[1] http://pypi.python.org/pypi/django-floppyforms
[2] http://www.google-melange.com/gsoc/project/google/gsoc2011/gregmuellegger/5001

django-crispy-forms

django-crispy-forms [3] is about form layouts too. The main part is in Python code: helpers and layout classes (configuration).

It could be an interesting alternative when you don't mind the layouts to be configured in Python code, i.e. when your web designers can contribute to Python code or when your developers do the web design.

More?

Of course there are other projects! As an example, around the Gsoc2011, some "proof-of-concept" projects appear.

You can find some of them at http://www.djangopackages.com/grids/g/forms/

References

2.8.2 License

2.8.3 Authors

- Benoît Bryon <benoit@marmelune.net>

[3] http://pypi.python.org/pypi/django-crispy-forms/1.1.4

2.8.4 Changes

0.3 (unreleased)

- Stopped development of django-formrenderingtools. Django-floppyforms is the recommended alternative.
- Cleaned and simplified project layout.

0.2.3 (2012-07-25)

- Fixed packaging: VERSION file was missing in distribution.

0.2.2 (2012-07-25)

- Fixed issue #27: templates are distributed with the package.
- Improved documentation.
- Improved development process.

0.2.1 (2011-05-22)

- Python module rename: from django-formrenderingtools to djc.formrenderingtools. The package name does not change (still django-formrenderingtools on Pypi).
- Improved documentation
- Improved packaging. Buildout integration.
- A demo project is now part of the sourcecode, so that one can quickly discover the application and perform experiments in a sandbox.
- Replaced dependency of Django >= 1.1 by Django >= 1.0

0.1 (2010-06-03)

- Initial release as a package
- Release on PyPI.

0.0 (2008-10-14)

- Initial commit.
- Closed-source and not packaged at the beginning.

CHAPTER 3

Credits and license

This application is published under the BSD license. See License and Authors for details.