formulation Documentation

Release 2.0.11

Curtis Maloney

November 11, 2014

Contents

Putting form rendering in its place.

Contents:

CHAPTER 1

Template Tags

Formulation works by providing a number of template tags.

1.1 The `form` tag

The `form` tag loads the template, and puts its blocks in a dict in the context, called *formulation*. You typically won't access this directly, as it's raw BlockNode instances.

```
{% form "widgets/bootstrap.form" %}
...
{% endform %}
```

You can optionally pass the form you will be using, also. This will allow the `field` tag to reference fields by name, instead of instance.

1.1.1 Template inheritance

Widget templates are just normal templates, so {% extends %} still works as expected. This lets you define a base, common form template, and localised extensions where you need.

1.2 The `field` tag

Used to render a form field, optionally specifying the widget to use.

```
{% field formfield [widget name] [key=value...] %}
```

You can think of the field tag as being like *{% include %}* but for blocks. However, it also adds many attributes from the form field into the context.

1.2.1 Values from `BoundField`

The following values are take from the `BoundField`:

- css_classes
- errors
- field

- form
- help_text
- html_name
- id_for_label
- label
- name
- value

1.2.2 Values from `Field`

And these from the `Field` itself:

- choices
- widget
- required

Any extra keyword arguments you pass to the field tag will overwrite values of the same name.

1.2.3 Auto-widget

If you omit the widget in the {% field %} tag, formulation will try to auto-detect the block to use. It does so by looking for the first block to match one of the following patterns:

- {field}_{widget}_{name}
- {field}_{name}
- {widget}_{name}
- {field}_{widget}
- {name}
- {widget}
- {field}

Where 'field' is the form field class (e.g. CharField, ChoiceField, etc), 'widget' is the widget class name (e.g. NumberInput, DateTimeInput, etc) and 'name' is the name of the field.

If no block is found, a TemplateSyntaxError is raised.

1.3 The `use` tag

You may have some chunks of templating that aren't fields, but are useful within the form. For these, write them as blocks in your *xyz.form* template, then use the *{% use %}* to include them:

1.3.1 demo.html

```
{% form "demo.form" %}
...
{% use "actions" submit="Update" %}
{% endform %}
```

1.3.2 demo.form

```
{% block actions %}
<div class="actions">
    <input type="submit" value="{{ submit|default:"Save" }}">
    <a href="/">Cancel</a>
</div>
{% endblock %}
```

It works just like include, but will use a block from the current widget template.

CHAPTER 2

Templates

Formulation ships with a sample template which tries to emulate the default Django form rendering as closely as possible.

The base field (called "input") looks like this:

```
{% block input %}
{% use "_label" %}
{% with field_type=field_type|default:"text" %}
<input type="{{ field_type }}"
    name="{{ html_name }}"
    id="{{ id }}"
    value="{{ value|default:"" }}"
    class="{{ css_classes }} {{ errors|yesno:"error," }}"
    {{ widget.attrs|flat_attrs }}
    {{ required|yesno:"required," }}
    {{ autofocus|yesno:"autofocus," }}
    {% if placeholder %}placeholder="{{ placeholder }}"{% endif %}
>
{% endwith %}
{% use "_help" %}
{% use "_errors" %}
{% endblock %}
```

There are 3 supplementary blocks it uses, making it easier for you to customise rendering without having to rewrite the whole template.

```
{% block _label %}
{% if label %}<label id="{{ id_for_label }}" for="{{ id }}">{{ label }}</label>{% endif %}
{% endblock %}

{% block _help %}
{{ help_text }}
{% endblock %}

{% block _errors %}
{% if errors %}
<ul class="errorlist">
{% for error in errors %}
    <li class="error">{{ error }}</li>
{% endfor %}
</ul>
{% endif %}
{% endblock %}
```

2.1 Examples

It can be helpful to look at how some of the default widgets are implemented to see how simple it can be.

```
{% block TextInput %}{% use "input" %}{% endblock %}
```

The basic TextInput uses the input widget without any alterations.

```
{% block EmailInput %}{% use "input" field_type="email" %}{% endblock %}
```

The EmailInput simply provides an override for field_type.

```
{% block PasswordInput %}{% use "input" field_type="password" value="" %}{% endblock %}
```

PasswordInput ensures the value is blanked out.

```
{% block DateInput %}{% use "input" field_type="date" value=value|date:'Y-m-d' %}{% endblock %}
```

DateInput, as well as DateTimeInput and TimeInput, use the `date` filter to convert the value to a userful format.

CHAPTER 3

Extras

3.1 The `flat_attrs` filter

This is simply a wrapper around `django.forms.utils.flatatt()`

It converts a dict of attributes into a string, in proper key=”value” syntax. The values will be escaped, but keys will not.

3.2 The `reuse` tag

There is also the `{% reuse %}` template tag, which allows you to reuse any template block within the current template [as opposed to the form widget template] like a macro. Again, it follows the same syntax as the `{% include %}` tag:

```
{% load reuse %}
{% reuse "otherblock" foo=1 %}
```

You can also pass a list of block names to search; first found will be used.

Note: `reuse` can only be used in templates which `{% extend %}` another template.

3.3 Using `use` for macros

Formulation is not limited to forms and fields. There’s no reason you can’t also use it to abstract commonly used fragments of template code.

```
{% form "widgets.form" %}

{% use "framed-box" title="Some box!" %}

...

{% endform %}
```

CHAPTER 4

Thanks

- kezabelle for the name
- bradleyayers for ideas on supporting multiple fields. (now removed)
- SmileyChris for the idea to “explode” fields into the context
- jwa for major testing and bug hunting
- schinkel for packaging help
- mbrochh for inspiring the name lookup idea
- Sergei Maertens for helping fix the leaky render context

CHAPTER 5

Changelog

5.1 v2.0.11

Bugs Fixed:

- Use the “new” method of request_context instead of deep copy. [Fixes #23]
- Refactor tests to make easier to run

5.2 v2.0.10

Bugs Fixed:

- One more change to BlockContext handling. [More thanks to Sergei Maertens]

5.3 v2.0.9

Bugs Fixed:

- Fix dirty BlockContext issue introduced in 2.0.8 [Thanks Sergei Maertens]
- Removed undocumented render_form

5.4 v2.0.8

Bugs Fixed:

- Ensure value is a comparable type in choices widgets
- Fixed default widget for select types to include display string
- Allow {{ block.super }} to work

5.5 v2.0.7.1

Bugs Fixed:

- Change list() to [] to not turn strings into lists

5.6 v2.0.7

Bugs Fixed:

- Fixed renamed variables in reuse tag
- Fixed testing current value in Select widget template
- Fixed value in Checkbox widget template
- force_text on choices values

Enhancements:

- Improved documentation
- Improved test coverage

Thanks to jwa

5.7 v2.0.6

Bugs Fixed:

- Removed duplicate EmailField block

Enhancements:

- Changed to using contextlib
- Allow a list of block names to be passed to {% reuse %}
- Added sphinx docs
- Added field lookup by name

5.8 v2.0.5

Bugs Fixed:

- Packaging fix

Enhancements:

- Improved docs
- Added {% render_form %} tag

5.9 v2.0.4

Bugs Fixed:

- Fixed date/time formatting in default template

5.10 v2.0.3

Bugs Fixed:

- Added tests (thanks jwa!)
- Fixed auto widget (thanks jwa!)

Enhancements:

- Improved templates (thanks jwa!)
- Began Py3 compatibility (thanks jwa!)

5.11 v2.0.2

Bugs Fixed:

- Fix importing of form.util(s) to make Django 1.5 compatible

5.12 v2.0.1

Bugs Fixed:

- Fixed context over-stacking (#5)

Enhancements:

- Added `flat_attrs` filter
- Changed default template to include templates for all stock Django widgets

5.13 v2.0.0

Enhancements:

- Changed to explode field and widget attributes into the context

CHAPTER 6

Overview

It's fairly well accepted, now, that having the form rendering decisions in your code is less than ideal.

However, most template-based solutions wind up being slow, because they rely on many templates per form.

Formulation works by defining all the widgets for your form in a single "widget template", and loading it once for the form.

CHAPTER 7

Installation

You can install formulation using:

```
$ pip install formulation
```

You will need to add *'formulation'* to your *settings.INSTALLED_APPS*.

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*