

---

# **crispy-form-foundation Documentation**

***Release 0.3.7.1***

**David THENON**

November 15, 2014



<b>1</b>	<b>Links</b>	<b>3</b>
<b>2</b>	<b>Requires</b>	<b>5</b>
2.1	Table of contents . . . . .	5
	<b>Python Module Index</b>	<b>11</b>



This is a [Django](#) application to add [django-crispy-forms](#) layout objects for [Foundation](#).

This app does not embed a [Foundation](#) release, you will have to install it yourself.



---

### Links

---

- Read the documentation on [Read the docs](#);
- Download his [PyPi package](#);
- Clone it on his [Github repository](#);
- Demo app : [crispy-forms-foundation-demo](#);





---

## Requires

---

- `django-crispy-forms = 1.4.x`;

## 2.1 Table of contents

### 2.1.1 Install

Register the app in your project settings like that :

```
INSTALLED_APPS = (
    ...
    'crispy_forms',
    'crispy_forms_foundation',
    ...
)
```

Then append this part to specify usage of the Foundation set :

```
# Default layout to use with "crispy_forms"
CRISPY_TEMPLATE_PACK = 'foundation-5'
```

If not defined, the default template pack name used is `foundation-5`, also you can use `foundation-3` but pay attention that is not really maintained.

All other `django-crispy-forms` settings option apply, see its documentation for more details.

### 2.1.2 Usage

Import **crispy-forms-foundation** then you can use the layout objects in your form :

```
from crispy_forms_foundation.layout import Layout, Fieldset, Field, SplitDateTimeField, Row, RowFluid

class YourForm(forms.ModelForm):
    """
    *Page* form
    """
    def __init__(self, *args, **kwargs):
        self.helper = FormHelper()
        self.helper.form_action = '.'
        self.helper.layout = Layout(
            Fieldset(
```

```
        ugettext('Content'),
        'title',
        'content',
    ),
    Fieldset(
        ugettext('Display settings'),
        Row(
            Column('template', css_class='large-6'),
            Column('order', css_class='large-3'),
            Column('visible', css_class='large-3'),
        ),
    ),
    Fieldset(
        ugettext('Publish settings'),
        'parent',
        Row(
            Column(SplitDateTimeField('published', css_class='large-6'),
            Column('slug', css_class='large-6'),
        ),
    ),
    ButtonHolder(
        Submit('submit_and_continue', ugettext('Save and continue')),
        Submit('submit', ugettext('Save')),
    ),
)

super(YourForm, self).__init__(*args, **kwargs)
```

The embedded templates are in `crispy_forms_foundation/templates/foundation`.

## Layout items

Inherits from the “uni\_form” Layout objects to force templates on `TEMPLATE_PACK` and use of Foundation CSS classes

Also the templates are more clean than the included ones from `crispy_forms` which produce too much spaces and newlines in the final HTML.

**class** `crispy_forms_foundation.layout.Button` (*name, value, \*\*kwargs*)

Used to create a Submit input descriptor for the `{% crispy %}` template tag:

```
button = Button('Button 1', 'Press Me!')
```

---

**Note:** The first argument is also slugified and turned into the id for the button.

---

**class** `crispy_forms_foundation.layout.ButtonGroup` (*\*fields, \*\*kwargs*)

It wraps fields in a `<ul class="button-group">`

This is where you should put Layout objects that render to form buttons like `Submit`. It should only hold *HTML* and *BaseInput* inherited objects.

Example:

```
ButtonGroup(
    Submit('Save', 'Save'),
    Button('Cancel', 'Cancel'),
)
```

**class** `crispy_forms_foundation.layout.ButtonHolder(*fields, **kwargs)`

It wraps fields in a `<div class="button-holder">`

This is where you should put Layout objects that render to form buttons like Submit. It should only hold HTML and BaseInput inherited objects.

Example:

```
ButtonHolder(
    HTML(<span style="display: hidden;">Information Saved</span>),
    Submit('Save', 'Save')
)
```

**class** `crispy_forms_foundation.layout.ButtonHolderPanel(field, *args, **kwargs)`

Act like ButtonHolder but add a panel css class on the main div

**class** `crispy_forms_foundation.layout.Column(field, *args, **kwargs)`

It wraps fields in a div. If not defined, CSS class will default to `large-12` columns. `columns` class is always appended, so you don't need to specify it.

This is the column from the [Foundation Grid](#), all columns should be contained in a **Row** or a **RowFluid** and you will have to define the column type in the `css_class` attribute.

Example:

```
Column('form_field_1', 'form_field_2', css_class='small-12 large-6')
```

Will render to something like that:

```
<div class="small-12 large-6 columns">...</div>
```

`columns` class is always appended, so you don't need to specify it.

If not defined, `css_class` will default to `large-12`.

**class** `crispy_forms_foundation.layout.Div(*fields, **kwargs)`

It wraps fields in a `<div>`

You can set `css_id` for a DOM id and `css_class` for a DOM class. Example:

```
Div('form_field_1', 'form_field_2', css_id='div-example', css_class='divs')
```

**class** `crispy_forms_foundation.layout.Field(*args, **kwargs)`

Layout object, It contains one field name, and you can add attributes to it easily. For setting class attributes, you need to use `css_class`, as `class` is a Python keyword.

Example:

```
Field('field_name', style="color: #333;", css_class="whatever", id="field_name")
```

**class** `crispy_forms_foundation.layout.Fieldset(legend, *fields, **kwargs)`

It wraps fields in a `<fieldset>`:

```
Fieldset("Text for the legend",
    'form_field_1',
    'form_field_2'
)
```

The first parameter is the text for the fieldset legend. This text is context aware, so you can do things like :

```
Fieldset("Data for {{ user.username }}",
    'form_field_1',
```

```
        'form_field_2'
    )
```

**class** `crispy_forms_foundation.layout.Hidden(name, value, **kwargs)`

Used to create a Hidden input descriptor for the `{% crispy %}` template tag.

**class** `crispy_forms_foundation.layout.InlineField(field, label_column='large-3', input_column='large-9', label_class='', *args, **kwargs)`

Layout object for rendering an inline field with Foundation

Example:

```
InlineField('field_name')
```

**class** `crispy_forms_foundation.layout.InlineJustifiedField(field, *args, **kwargs)`

Same as `InlineField` but default is to be right aligned with a vertical padding

**class** `crispy_forms_foundation.layout.MultiField(label, *fields, **kwargs)`

`MultiField` container. Renders to a `MultiField`

**class** `crispy_forms_foundation.layout.Panel(field, *args, **kwargs)`

Act like `Div` but add a panel css class.

Example:

```
Panel('form_field_1', 'form_field_2', css_id='div-example', css_class='divs')
```

**class** `crispy_forms_foundation.layout.Reset(name, value, **kwargs)`

Used to create a Reset button input descriptor for the `{% crispy %}` template tag:

```
reset = Reset('Reset This Form', 'Revert Me!')
```

---

**Note:** The first argument is also slugified and turned into the id for the reset.

---

**class** `crispy_forms_foundation.layout.Row(*fields, **kwargs)`

It wraps fields in a div whose default class is `row`. Example:

```
Row('form_field_1', 'form_field_2', 'form_field_3')
```

Act as a div container row, it will embed its items in a div like that:

```
<div class="row">Your stuff</div>
```

**class** `crispy_forms_foundation.layout.RowFluid(*fields, **kwargs)`

It wraps fields in a div whose default class is `"row row-fluid"`. Example:

```
RowFluid('form_field_1', 'form_field_2', 'form_field_3')
```

It has a same behaviour than `Row` but add a CSS class `"row-fluid"` that you can use to have top level row that take all the container width. You have to put the CSS for this class to your CSS stylesheets. It will embed its items in a div like that:

```
<div class="row row-fluid">Your stuff</div>
```

The CSS to add should be something like that:

```
.row-fluid {
    width: 100%;
    max-width: 100%;
```

```

    min-width: 100%;
}

```

**class** `crispy_forms_foundation.layout.SplitDateTimeField(*args, **kwargs)`

Just an inherit from `crispy_forms.layout.Field` to have a common Field for displaying field with the `django.forms.extra.SplitDateTimeWidget` widget.

Simply use a specific template

**class** `crispy_forms_foundation.layout.Submit(name, value, **kwargs)`

Used to create a Submit button descriptor for the `{% crispy %}` template tag:

```
submit = Submit('Search the Site', 'search this site')
```

---

**Note:** The first argument is also slugified and turned into the id for the submit button.

---

**class** `crispy_forms_foundation.layout.SwitchField(field, *args, **kwargs)`

A specific field to use Foundation form switches

You should only use this with a checkbox field

Example:

```
SwitchField('field_name', style="color: #333;", css_class="whatever", id="field_name")
```

## Use Foundation 5 Abide

You can use [Abide](#) validation in your form but note that there is no support within the layout objects. You will have to add the `required` attribute (and eventually its pattern) on your field widgets in your form.

So to enable [Abide](#) you'll have to load its Javascript library if you don't load yet the whole Foundation library, then in your form helper you will have to its attribute on the form like this :

```

class SampleForm(forms.Form):
    def __init__(self, *args, **kwargs):
        self.helper = FormHelper()
        self.helper.attrs = {'data_abide': ''}
        self.helper.form_action = '.'
        self.helper.layout = Layout(
            ...
        )

    super(SampleForm, self).__init__(*args, **kwargs)

```

Then add the required attribute on a field widget like this :

```
textarea_input = forms.CharField(label=_('Textarea'), widget=forms.Textarea(attrs={'required': ''}), ...)
```

## 2.1.3 Changelog

### Version 0.3.6

- Add **'ButtonGroup'** to use Foundation's Button groups instead of Button holder;
- Add **'Panel'** layout element that act like a Div but add a panel css class name;

### Version 0.3.5

- Add **'SwitchField' \_** field;

### Version 0.3.3

- Fix bad template includes in some templates;

### Version 0.3.2

- Fix some css class in templates;
- Add documentation for **'Abide' \_** usage;
- Add **'ButtonHolderPanel' \_** layout object;

### Version 0.3.1

- Added **'InlineField' \_** and **'InlineJustifiedField' \_**;

### Version 0.3.0

Some backward incompatible change have been done, be sure to check them before upgrading.

- Removed sample view, url and templates. If needed you can find a Django app sample on **'crispy-forms-foundation-demo' \_**;
- Moving foundation template pack name and its directory to foundation-3. You have to change your `settings.CRISPY_TEMPLATE_PACK` if you used the old one;
- Add foundation-5 template pack, it is now the default template pack;
- Removing camelcase on some css classes :
  - `ctrlHolder` has changed to `holder`;
  - `buttonHolder` has changed to `button-holder`;
  - `asteriskField` has changed to `asterisk`;
  - `errorField` has changed to `error`;
  - `formHint` has changed to `hint`;
  - `inlineLabel` has changed to `inline-label`;
  - `multiField` has changed to `multiple-fields`;

## C

`crispy_forms_foundation.layout`, [6](#)





## B

Button (class in `crispy_forms_foundation.layout`), 6  
ButtonGroup (class in `crispy_forms_foundation.layout`), 6  
ButtonHolder (class in `crispy_forms_foundation.layout`), 6  
ButtonHolderPanel (class in `crispy_forms_foundation.layout`), 7

## C

Column (class in `crispy_forms_foundation.layout`), 7  
`crispy_forms_foundation.layout` (module), 6

## D

Div (class in `crispy_forms_foundation.layout`), 7

## F

Field (class in `crispy_forms_foundation.layout`), 7  
Fieldset (class in `crispy_forms_foundation.layout`), 7

## H

Hidden (class in `crispy_forms_foundation.layout`), 8

## I

InlineField (class in `crispy_forms_foundation.layout`), 8  
InlineJustifiedField (class in `crispy_forms_foundation.layout`), 8

## M

MultiField (class in `crispy_forms_foundation.layout`), 8

## P

Panel (class in `crispy_forms_foundation.layout`), 8

## R

Reset (class in `crispy_forms_foundation.layout`), 8  
Row (class in `crispy_forms_foundation.layout`), 8  
RowFluid (class in `crispy_forms_foundation.layout`), 8

## S

SplitDateTimeField (class in `crispy_forms_foundation.layout`), 9  
Submit (class in `crispy_forms_foundation.layout`), 9  
SwitchField (class in `crispy_forms_foundation.layout`), 9