crispy-form-foundation Documentation

Release 0.5.5

David THENON

February 01, 2017

Contents

This is a Django application to add django-crispy-forms layout objects for Foundation for sites.

This app does not include a Foundation for sites release, you will have to install it yourself in your projects.

CHAPTER 1

Links

- Read the documentation on Read the docs;
- Download his PyPi package;
- Clone it on his Github repository;

CHAPTER 2

Requires

- Django >=1.8, <=1.10;
- django-crispy-forms >= 1.4.x;

2.1 Table of contents

2.1.1 Install

1. Get it from PyPi:

```
pip install crispy-forms-foundation
```

2. Register app in your project settings:

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

3. Import default settings at the end of the settings file:

```
from crispy_forms_foundation.settings import *
```

Default template pack name used will be `foundation-5`.

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, SplitDateTimeField, Row, Column, ButtonH

class YourForm(forms.ModelForm):
    def __init__(self, *args, **kwargs):
        self.helper = FormHelper()
        self.helper.form_action = '.'
        self.helper.layout = Layout(
            Fieldset(
                'Content',
                'title',
                'content',
            ),
            Fieldset(
                'Display settings',
                Row(
                    Column('template', css_class='large-6'),
                    Column('order', css_class='large-3'),
                    Column('visible', css_class='large-3'),
                ),
            ),
            Fieldset(
                'Publish settings',
                'parent',
                Row(
                    Column(SplitDateTimeField('published'), css_class='large-6'),
                    Column('slug', css_class='large-6'),
                ),
            ),
            ButtonHolder(
                Submit('submit_and_continue', 'Save and continue'),
                Submit('submit', 'Save'),
            ),
        )

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

The embedded templates are in `crispy_forms_foundation/templates/foundation-5`.

Use Foundation Abide validation

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 validation pattern) on your field widgets in your form like this:

```
title = forms.CharField(label=_('Title'), widget=forms.TextInput(attrs={'required':''}), required=Tru
```

To enable Abide on your form, 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 add its attribute on the form like this :

```
class SampleForm(forms.Form):
    title = forms.CharField(label=_('Title'), widget=forms.TextInput(attrs={'required':''}), require
    textarea_input = forms.CharField(label=_('Textarea'), widget=forms.Textarea(attrs={'required':''

    def __init__(self, *args, **kwargs):
        self.helper = FormHelper()

        # Enable Abide validation on the form
        self.helper.attrs = {'data_abide': ''}

        self.helper.form_action = '.'
        self.helper.layout = Layout(
            ...
        )

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

You can also set an Abide error message directly on the field like this :

```
class SampleForm(forms.Form):
    def __init__(self, *args, **kwargs):
        super(SampleForm, self).__init__(*args, **kwargs)
        self.fields['textarea_input'].abide_msg = "This field is required !"
```

Support within tabs

Default Abide behavior is not aware of Tabs and so input errors can be hided when they are not in the active tab.

crispy-forms-foundation ships a jQuery plugin that add support for this usage, you will need to load it in your pages then initialize it on your form:

```
<script type="text/javascript" src="{{ STATIC_URL }}js/crispy_forms_foundation/plugins.js"></script>
<script type="text/javascript">
//<![CDATA[
$(document).ready(function() {
    $('form').abide_support_for_tabs();
});
//]]>
</script>
```

This way, all input errors will be raised to their tab name that will display an error mark.

Support within accordions

Like with tabs, there is a jQuery plugin to add Abide support within accordions.

You will need to load it in your pages then initialize it on your form:

```
<script type="text/javascript" src="{{ STATIC_URL }}js/crispy_forms_foundation/plugins.js"></script>
<script type="text/javascript">
//<![CDATA[
$(document).ready(function() {
    $('form').abide_support_for_accordions();
});
//]]>
</script>
```

Automatic form layout

There is some forms you can use to quickly and automatically create a Foundation layout for your forms. This is mostly for fast integration or prototyping because it will probably never totally fit to your design.

class `crispy_forms_foundation.forms.`**`FoundationForm`**(**args*, ***kwargs*)

Bases: *crispy_forms_foundation.forms.FoundationFormMixin*, `django.forms.forms.Form`

A **Django form** that inherit from `FoundationFormMixin` to automatically build a form layout

Example:

```
from django import forms
from crispy_forms_foundation.forms import FoundationForm

class YourForm(FoundationForm):
    title = "Testing"
```

```
        action = 'test'
        layout = Layout(Fieldset("Section", "my_field", "my_field_2"))
        switches = False
        attrs = {'data_abide': ""}

        title = forms.CharField(label='Title', required=True)
        slug = forms.CharField(label='Slug', required=False)
```

class `crispy_forms_foundation.forms.`**`FoundationFormMixin`**

Bases: `object`

Mixin to implement the layout helper that will automatically build a form layout

Generally, you will prefer to use `FoundationForm` or `FoundationModelForm` instead.

If you still want to directly use this mixin you'll just have to execute `FoundationFormMixin.init_helper()` in your form init.

Attributes

title If set, defines the form's title

layout If set, override the default layout for the form

error_title Defines the error title for non field errors

form_id Defines the id of the form

classes Defines the classes used on the form

action Defines the action of the form. `reverse` will be called on the value. On failure the value will be assigned as is

method Defines the method used for the action

attrs Defines the attributes of the form

switches If True, will replace all fields checkboxes with switches

submit Adds a submit button on the form. Can be set to a Submit object or a string which will be used as the value of the submit button

title_templatestring Template string used to display form title (if any)

class `crispy_forms_foundation.forms.`**`FoundationModelForm`**`(`**args*, ***kwargs*`)`

Bases: *`crispy_forms_foundation.forms.FoundationFormMixin`*, `django.forms.models.ModelForm`

A **Django Model form** that inherit from `FoundationFormMixin` to automatically build a form layout

Example:

```
    from crispy_forms_foundation.forms import FoundationModelForm

    class YourForm(FoundationModelForm):
        title = "Testing"
        action = 'test'
        layout = Layout(Fieldset("Section", "my_field", "my_field_2"))
        switches = False
        attrs = {'data_abide': ""}

        class Meta:
            model = MyModel
            fields = ['my_field', 'my_field_2', 'my_field_3']
```

2.1.3 Layout items

Layout items for Foundation components

Inherits from the default **crispy_forms** layout objects to force templates on the right `TEMPLATE_PACK` (defined from `settings.CRISPY_TEMPLATE_PACK`) and implements Foundation components.

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

Bases: `crispy_forms.layout.Div`

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.`**`Panel`**(*field*, **args*, ***kwargs*)

Bases: `crispy_forms.layout.Div`

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.`**`Row`**(**fields*, ***kwargs*)

Bases: *`crispy_forms_foundation.layout.base.Div`*

Wrap 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">Content</div>
```

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

Bases: *`crispy_forms_foundation.layout.grid.Row`*

Wrap 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">Content</div>
```

The CSS to add should be something like that:

```
/*
* Fluid row takes the full width but keep normal row and columns
* behaviors
*/
@mixin row-fluid-mixin {
    max-width: 100%;
    // Restore the initial behavior restrained to the grid
    .row{
        margin: auto;
        @include grid-row;
        // Preserve nested fluid behavior
        &.row-fluid{
            max-width: 100%;
        }
    }
}
.row.row-fluid{
    @include row-fluid-mixin;
}
@media #{$small-up} {
    .row.small-row-fluid{ @include row-fluid-mixin; }
}
@media #{$medium-up} {
    .row.medium-row-fluid{ @include row-fluid-mixin; }
}
@media #{$large-up} {
    .row.large-row-fluid{ @include row-fluid-mixin; }
}
@media #{$xlarge-up} {
    .row.xlarge-row-fluid{ @include row-fluid-mixin; }
}
@media #{$xxlarge-up} {
    .row.xxlarge-row-fluid{ @include row-fluid-mixin; }
}
```

It must be included after Foundation grid component is imported.

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

Bases: *`crispy_forms_foundation.layout.base.Div`*

Wrap 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.`**`Field`**(**args*, ***kwargs*)

Bases: `crispy_forms.layout.Field`

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

Example:

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

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

MultiField container. Render to a MultiField

class `crispy_forms_foundation.layout.`**`SplitDateTimeField`**(**args*, ***kwargs*)
Bases: *`crispy_forms_foundation.layout.fields.Field`*

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

Layout object for rendering an inline field with Foundation

Example:

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

Or:

```
InlineField('field_name', label_column='large-8',
            input_column='large-4', label_class='')
```

`label_column`, `input_column`, `label_class`, are optional argument.

class `crispy_forms_foundation.layout.`**`InlineJustifiedField`**(*field*, **args*, ***kwargs*)
Bases: *`crispy_forms_foundation.layout.fields.InlineField`*

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

class `crispy_forms_foundation.layout.`**`SwitchField`**(*field*, **args*, ***kwargs*)
Bases: `crispy_forms.layout.Field`

A specific field to use Foundation form switches

You must only use this with a checkbox field and this is a *raw* usage of this Foundation element, you should see `InlineSwitchField` instead.

Example:

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

class `crispy_forms_foundation.layout.`**`InlineSwitchField`**(*field*, **args*, ***kwargs*)
Bases: *`crispy_forms_foundation.layout.fields.InlineField`*

Like `SwitchField` it use Foundation form switches with checkbox field but within an `InlineField`

Contrary to `SwitchField` this play nice with the label to be able to display it (as Foundation form switches default behavior is to hide the label text)

Example:

```
InlineSwitchField('field_name')
```

Or:

```
InlineSwitchField('field_name', label_column='large-8',
                  input_column='large-4', label_class='',
                  switch_class="inline")
```

`label_column`, `input_column`, `label_class`, `switch_class` are optional argument.

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

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*)
Bases: *`crispy_forms_foundation.layout.buttons.ButtonHolder`*

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

class `crispy_forms_foundation.layout.`**`ButtonGroup`**(**fields*, ***kwargs*)
Bases: `crispy_forms.layout.LayoutObject`

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.`**`Button`**(*name*, *value*, ***kwargs*)
Bases: `crispy_forms.layout.BaseInput`

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.`**`Submit`**(*name*, *value*, ***kwargs*)
Bases: `crispy_forms.layout.BaseInput`

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.`**`Hidden`**`(`*name*, *value*, ***kwargs*`)`
Bases: `crispy_forms.layout.Hidden`

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

class `crispy_forms_foundation.layout.`**`Reset`**`(`*name*, *value*, ***kwargs*`)`
Bases: `crispy_forms.layout.BaseInput`

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.`**`Fieldset`**`(`*legend*, **fields*, ***kwargs*`)`
Bases: `crispy_forms.layout.Fieldset`

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.`**`TabItem`**`(`*name*, **fields*, ***kwargs*`)`
Bases: `crispy_forms.bootstrap.Tab`

Tab item object. It wraps fields in a div whose default class is "tabs" and takes a name as first argument.

The item name is also slugified to build an id for the tab if you don't define it using `css_id` argument.

Example:

```
TabItem('My tab', 'form_field_1', 'form_field_2', 'form_field_3')
```

`TabItem` layout item has no real utility out of a `TabHolder`.

`has_errors``(`*form*`)`
Find tab fields are listed as invalid

`render_link``(`*form*`)`
Render the link for the tab-pane. It must be called after render so `css_class` is updated with `active` class name if needed.

class `crispy_forms_foundation.layout.`**`TabHolder`**`(`**fields*, ***kwargs*`)`
Bases: `crispy_forms.bootstrap.TabHolder`

Tabs holder object to wrap Tab item objects in a container:

```
TabHolder(
    TabItem('My tab 1', 'form_field_1', 'form_field_2'),
    TabItem('My tab 2', 'form_field_3')
)
```

`TabHolder` direct children should allways be a `TabItem` layout item.

The first `TabItem` containing a field error will be marked as *active* if any, else this will be just the first `TabItem`.

class `crispy_forms_foundation.layout.`**`VerticalTabHolder`**(**fields*, ***kwargs*)
Bases: *`crispy_forms_foundation.layout.containers.TabHolder`*

VerticalTabHolder appends vertical class to TabHolder container

class `crispy_forms_foundation.layout.`**`AccordionItem`**(*name*, **fields*, ***kwargs*)
Bases: `crispy_forms.bootstrap.AccordionGroup`

Accordion item object. It wraps given fields inside an accordion tab. It takes accordion tab name as first argument.

The item name is also slugified to build an id for the tab if you don't define it using `css_id` argument.

Example:

```
AccordionItem("group name", "form_field_1", "form_field_2")
```

class `crispy_forms_foundation.layout.`**`AccordionHolder`**(**fields*, ***kwargs*)
Bases: `crispy_forms.bootstrap.Accordion`

Accordion items holder object to wrap Accordion item objects in a container:

```
AccordionHolder(
    AccordionItem("group name", "form_field_1", "form_field_2"),
    AccordionItem("another group name", "form_field"),
)
```

`AccordionHolder` direct children should allways be a `AccordionItem` layout item.

The first `AccordionItem` containing a field error will be marked as *active* if any, else this will be just the first `AccordionItem`.

Base

Basic layout items

class `crispy_forms_foundation.layout.base.`**`Div`**(**fields*, ***kwargs*)
Bases: `crispy_forms.layout.Div`

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.base.`**`Panel`**(*field*, **args*, ***kwargs*)
Bases: `crispy_forms.layout.Div`

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

Example:

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

Fields

Field layout items

See :

- Foundation forms for input field components;
- Foundation Switches for switches components;

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

Bases: `crispy_forms.layout.Field`

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

Example:

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

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

Bases: *`crispy_forms_foundation.layout.fields.Field`*

Layout object for rendering an inline field with Foundation

Example:

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

Or:

```
InlineField('field_name', label_column='large-8',
            input_column='large-4', label_class='')
```

`label_column`, `input_column`, `label_class`, are optional argument.

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

Bases: *`crispy_forms_foundation.layout.fields.InlineField`*

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

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

Bases: *`crispy_forms_foundation.layout.fields.InlineField`*

Like `SwitchField` it use Foundation form switches with checkbox field but within an `InlineField`

Contrary to `SwitchField` this play nice with the label to be able to display it (as Foundation form switches default behavior is to hide the label text)

Example:

```
InlineSwitchField('field_name')
```

Or:

```
InlineSwitchField('field_name', label_column='large-8',
                  input_column='large-4', label_class='',
                  switch_class="inline")
```

`label_column`, `input_column`, `label_class`, `switch_class` are optional argument.

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

MultiField container. Render to a MultiField

class `crispy_forms_foundation.layout.fields.`**`SplitDateTimeField`**(**args*, ***kwargs*)
Bases: *`crispy_forms_foundation.layout.fields.Field`*

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.fields.`**`SwitchField`**(*field*, **args*, ***kwargs*)
Bases: `crispy_forms.layout.Field`

A specific field to use Foundation form switches

You must only use this with a checkbox field and this is a *raw* usage of this Foundation element, you should see `InlineSwitchField` instead.

Example:

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

Grid

Foundation grid layout objects

See Foundation Grid for grid components.

class `crispy_forms_foundation.layout.grid.`**`Column`**(*field*, **args*, ***kwargs*)
Bases: *`crispy_forms_foundation.layout.base.Div`*

Wrap 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.grid.`**`Row`**(**fields*, ***kwargs*)
Bases: *`crispy_forms_foundation.layout.base.Div`*

Wrap 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">Content</div>
```

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

Bases: *`crispy_forms_foundation.layout.grid.Row`*

Wrap 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">Content</div>
```

The CSS to add should be something like that:

```
/*
* Fluid row takes the full width but keep normal row and columns
* behaviors
*/
@mixin row-fluid-mixin {
    max-width: 100%;
    // Restore the initial behavior restrained to the grid
    .row{
        margin: auto;
        @include grid-row;
        // Preserve nested fluid behavior
        &.row-fluid{
            max-width: 100%;
        }
    }
}
.row.row-fluid{
    @include row-fluid-mixin;
}
@media #{$small-up} {
    .row.small-row-fluid{ @include row-fluid-mixin; }
}
@media #{$medium-up} {
    .row.medium-row-fluid{ @include row-fluid-mixin; }
}
@media #{$large-up} {
    .row.large-row-fluid{ @include row-fluid-mixin; }
}
@media #{$xlarge-up} {
    .row.xlarge-row-fluid{ @include row-fluid-mixin; }
}
@media #{$xxlarge-up} {
    .row.xxlarge-row-fluid{ @include row-fluid-mixin; }
}
```

It must be included after Foundation grid component is imported.

Buttons

Button layout items

See :

- Foundation buttons for button components;
- Foundation button groups for button groups components;

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

Bases: `crispy_forms.layout.BaseInput`

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.buttons.`**`ButtonGroup`**(**fields*, ***kwargs*)

Bases: `crispy_forms.layout.LayoutObject`

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.buttons.`**`ButtonHolder`**(**fields*, ***kwargs*)

Bases: `crispy_forms.layout.ButtonHolder`

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.buttons.`**`ButtonHolderPanel`**(*field*, **args*, ***kwargs*)

Bases: *`crispy_forms_foundation.layout.buttons.ButtonHolder`*

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

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

Bases: `crispy_forms.layout.Hidden`

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

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

Bases: `crispy_forms.layout.BaseInput`

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.buttons.`**`Submit`**(*name*, *value*, ***kwargs*)

Bases: `crispy_forms.layout.BaseInput`

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.

Form containers

Form container layout objects

See :

- Foundation forms for fieldset component;
- Foundation Accordion for accordion components;
- Foundation Tabs for tabs components;

class `crispy_forms_foundation.layout.containers.`**`AccordionHolder`**(**fields*, ***kwargs*)

Bases: `crispy_forms.bootstrap.Accordion`

Accordion items holder object to wrap Accordion item objects in a container:

```
AccordionHolder(
    AccordionItem("group name", "form_field_1", "form_field_2"),
    AccordionItem("another group name", "form_field"),
)
```

`AccordionHolder` direct children should allways be a `AccordionItem` layout item.

The first `AccordionItem` containing a field error will be marked as *active* if any, else this will be just the first `AccordionItem`.

class `crispy_forms_foundation.layout.containers.`**`AccordionItem`**(*name*, **fields*, ***kwargs*)

Bases: `crispy_forms.bootstrap.AccordionGroup`

Accordion item object. It wraps given fields inside an accordion tab. It takes accordion tab name as first argument.

The item name is also slugified to build an id for the tab if you don't define it using `css_id` argument.

Example:

```
AccordionItem("group name", "form_field_1", "form_field_2")
```

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

Bases: `crispy_forms.layout.Fieldset`

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.containers.`**`TabHolder`**(**fields*, ***kwargs*)

Bases: `crispy_forms.bootstrap.TabHolder`

Tabs holder object to wrap Tab item objects in a container:

```
TabHolder(
    TabItem('My tab 1', 'form_field_1', 'form_field_2'),
    TabItem('My tab 2', 'form_field_3')
)
```

`TabHolder` direct children should allways be a `TabItem` layout item.

The first `TabItem` containing a field error will be marked as *active* if any, else this will be just the first `TabItem`.

class `crispy_forms_foundation.layout.containers.`**`TabItem`**(*name*, **fields*, ***kwargs*)

Bases: `crispy_forms.bootstrap.Tab`

Tab item object. It wraps fields in a div whose default class is "tabs" and takes a name as first argument.

The item name is also slugified to build an id for the tab if you don't define it using `css_id` argument.

Example:

```
TabItem('My tab', 'form_field_1', 'form_field_2', 'form_field_3')
```

`TabItem` layout item has no real utility out of a `TabHolder`.

`has_errors`(*form*)

Find tab fields are listed as invalid

`render_link`(*form*)

Render the link for the tab-pane. It must be called after render so `css_class` is updated with `active` class name if needed.

class `crispy_forms_foundation.layout.containers.`**`VerticalTabHolder`**(**fields*, ***kwargs*)

Bases: *`crispy_forms_foundation.layout.containers.TabHolder`*

VerticalTabHolder appends vertical class to TabHolder container

2.2 History

2.2.1 Changelog

Version 0.5.5 - 2017/02/01

- Dropped support for Python 2.6 and Django<1.8;

- Added default app settings file;
- Added project test structure;
- Added pretty simple tests to cover layout elements which include some code;
- Added demo app taken from crispy-form-foundation-demo;
- Added dev and test requirements files;
- Updated setup.py;
- Added and enabled minified basic assets for Foundation 5 and 6 for test and demo;
- Finished demo urls/templates to work on every Foundation versions;
- Fixed Flake issues;
- Validated test with Tox for Python 2.7, Python 3.5 and Django>=1.8,<=1.10;

Everything should still work as with previous version.

Version 0.5.4

- Fixed `TabHolder` and `AccordionHolder` to have the right *active* behavior on their items: activate the first item with a field error if any, else just activate the first item;

Version 0.5.3

- Fixed bugs with button layout elements since django-crispy-forms==1.5.x, this is backward compatible with previous django-crispy-forms<1.5.x (with pull request #26 to close #25);
- Fixed package infos and README to be more explicit on Django compatibility (1.4 to 1.8 actually tested);

Version 0.5.2

- Use relative imports and enforce absolute imports;
- Add german and french translation with i18n;

Version 0.5.1

- Fix 'disable_csrf' option that was not honored in template forms;

Version 0.5.0

- Better layout elements organization;
- Merged pull request #20 for *Added Foundation tabs and accordion components based on crispy-forms bootstrap3 implementation*;
- Removed all stuff for Foundation 3 that is not supported anymore;
- Fix TabItem and TabHolder so tab inputs errors are raised to the Tab item;
- Fix AccordionItem and AccordionHolder so accordion inputs errors are raised to the accordion item name;
- Add jquery plugin to add Abide support within tabs and accordions so the input errors are raised to their title name and not hided into contents;

- Update documentation;

Version 0.4.1

- Added docs for submit button;
- Fixed bug where the class layout property was being used and modified by instances;
- Added Contributors to the doc;

Version 0.4

- Allow unicode characters in the form title in `forms.FoundationFormMixin`;
- Extended `forms.FoundationFormMixin.init_helper()` to allow more customization:
  - Renamed attribute input to submit as this is more descriptive
  - Allow to give a string which is used as display text for the Submit button
  - Allow to give a Submit instance wich is directly used
- Added `forms.FoundationFormMixin.title_templatestring` attribute to store template string used to display form title;
- Moved `forms.FoundationFormMixin.id` attribute name to `forms.FoundationFormMixin.form_id`;

Version 0.3.9

- Added `FoundationFormMixin`, `FoundationForm` and `FoundationModelForm` in `forms.py` to quickly and automatically create a Foundation layout;
- Added `InlineSwitchField` layout element for better switches usage;

Version 0.3.8

- Redesigned *non field errors*;
- Added abide error message on field;
- Added missing error message and help text on inline field;

Version 0.3.7

- Added better documentation with Sphinx in 'docs/';

Version 0.3.6

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

Version 0.3.5

- Added `SwitchField` field;

Version 0.3.3

- Fix bad template includes in some templates;

Version 0.3.2

- Fixed some css class in templates;
- Added documentation for `Abide` usage;
- Added `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;
- Moved `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;
- Added `foundation-5` template pack, it is now the default template pack;
- Removed 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`;

2.2.2 Contributors

- Philip Garnero (@PhilipGarnero);
- Juerg Rast (@jrast);
- JR (@jayarnielsen);
- Carsolcas (@carsolcas);
- Simon Bächler (@sbaechler);
- Manu Phatak (@bionikspoon);
- Florian Eßer (@flesser);

Index