WTForms-Django Documentation

Release 0.1

WTForms Team

December 25, 2015

Contents

Contents:

CHAPTER 1

WTForms-Django

This extension provides templatetags to make it easier to work with Django templates and WTForms' html attribute rendering. It also provides a generator for automatically creating forms based on Django ORM models.

1.1 Templatetags

Django templates does not allow arbitrarily calling functions with parameters, making it impossible to use the html attribute rendering feature of WTForms. To alleviate this, we provide a templatetag.

Adding *wtforms_django* to your INSTALLED_APPS will make the wtforms template library available to your application. With this you can pass extra attributes to form fields similar to the usage in jinja:

```
{% load wtforms %}

{% form_field form.username class="big_text" onclick="do_something()" %}
```

Note By default, using the `{{ form.field }}` syntax in django models will be auto-escaped. To avoid this happening, use Django's `{% autoescape off %}` block tag or use WTForms' *form_field* template tag.

1.2 Model forms

wtforms_django.orm.**model_form**(*model*, *base_class=Form*, *only=None*, *exclude=None*, *field_args=None*, *converter=None*)

Create a wtforms Form for a given Django model class:

```
from wtforms_django.orm import model_form
from myproject.myapp.models import User
UserForm = model_form(User)
```

Parameters

- **model** – A Django ORM model class
- **base_class** – Base form class to extend from. Must be a `wtforms.Form` subclass.
- **only** – An optional iterable with the property names that should be included in the form. Only these properties will have fields.
- **exclude** – An optional iterable with the property names that should be excluded from the form. All other properties will have fields.

- **field_args** – An optional dictionary of field names mapping to keyword arguments used to construct each field object.
- **converter** – A converter to generate the fields based on the model properties. If not set, `ModelConverter` is used.

model_form() attempts to glean as much metadata as possible from inspecting the model's fields, and will even attempt to guess at what validation might be wanted based on the field type. For example, converting an *EmailField* will result in a `StringField` with the `Email` validator on it. if the *blank* property is set on a model field, the resulting form field will have the `Optional` validator set.

Just like any other Form, forms created by model_form can be extended via inheritance:

```
UserFormBase = model_form(User)

class UserForm(UserFormBase):
    new_pass     = PasswordField('', [validators.optional(), validators.equal_to('confirm_pass')
    confirm_pass = PasswordField()
```

When combined with `form iteration`, model_form is a handy way to generate dynamic CRUD forms which update with added fields to the model. One must be careful though, as it's possible the generated form fields won't be as strict with validation as a hand-written form might be.

1.3 ORM-backed fields

While linking data to most fields is fairly easy, making drop-down select lists using django ORM data can be quite repetitive. To this end, we have added some helpful tools to use the django ORM along with wtforms.

class `wtforms_django.fields.`**QuerySetSelectField**(*default field args*, *queryset=None*, *get_label=None*, *allow_blank=False*, *blank_text=u''*)

Given a QuerySet either at initialization or inside a view, will display a select drop-down field of choices. The *data* property actually will store/keep an ORM model instance, not the ID. Submitting a choice which is not in the queryset will result in a validation error.

Specify *get_label* to customize the label associated with each option. If a string, this is the name of an attribute on the model object to use as the label text. If a one-argument callable, this callable will be passed model instance and expected to return the label text. Otherwise, the model object's *__str__* or *__unicode__* will be used.

If *allow_blank* is set to *True*, then a blank choice will be added to the top of the list. Selecting this choice will result in the *data* property being *None*. The label for the blank choice can be set by specifying the *blank_text* parameter.

```
class ArticleEdit(Form):
    title    = StringField()
    column   = QuerySetSelectField(get_label='title', allow_blank=True)
    category = QuerySetSelectField(queryset=Category.objects.all())

def edit_article(request, id):
    article = Article.objects.get(pk=id)
    form = ArticleEdit(obj=article)
    form.column.queryset = Column.objects.filter(author=request.user)
```

As shown in the above example, the queryset can be set dynamically in the view if needed instead of at form construction time, allowing the select field to consist of choices only relevant to the user.

class wtforms_django.fields.**ModelSelectField**(*default field args*, *model=None*, *get_label=''*, *allow_blank=False*, *blank_text=u''*)

Like a QuerySetSelectField, except takes a model class instead of a queryset and lists everything in it.

class wtforms_django.fields.**DateTimeField**(**args*, ***kwargs*)

Adds support for Django's timezone utilities. Requires Django >= 1.4

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

W

D

M

Q

W