WTForms-Appengine Documentation

Release 0.1

WTForms Team

December 16, 2015

Contents

Contents:

CHAPTER 1

WTForms-Appengine

WTForms-Appengine includes support for AppEngine fields as well as auto-form generation from models.

Note: WTForms-Appengine supports both *appengine.ext.db* and *appengine.ext.ndb* style models now, and there is some overlap between them. For the near future, we will continue to support both, but at some point will go to only supporting AppEngine for python 2.7 and drop support for ext.db models as well.

1.1 Model Forms

Form generation utilities for App Engine's `db.Model` class.

The goal of `model_form()` is to provide a clean, explicit and predictable way to create forms based on `db.Model` classes. No malabarism or black magic should be necessary to generate a form for models, and to add custom non-model related fields: `model_form()` simply generates a form class that can be used as it is, or that can be extended directly or even be used to create other forms using `model_form()`.

Example usage:

```
from google.appengine.ext import db
from tipfy.ext.model.form import model_form

# Define an example model and add a record.
class Contact(db.Model):
    name = db.StringProperty(required=True)
    city = db.StringProperty()
    age = db.IntegerProperty(required=True)
    is_admin = db.BooleanProperty(default=False)

new_entity = Contact(key_name='test', name='Test Name', age=17)
new_entity.put()

# Generate a form based on the model.
ContactForm = model_form(Contact)

# Get a form populated with entity data.
entity = Contact.get_by_key_name('test')
form = ContactForm(obj=entity)
```

Properties from the model can be excluded from the generated form, or it can include just a set of properties. For example:

```
# Generate a form based on the model, excluding 'city' and 'is_admin'.
ContactForm = model_form(Contact, exclude=('city', 'is_admin'))

# or...

# Generate a form based on the model, only including 'name' and 'age'.
ContactForm = model_form(Contact, only=('name', 'age'))
```

The form can be generated setting field arguments:

```
ContactForm = model_form(Contact, only=('name', 'age'), field_args={
    'name': {
        'label': 'Full name',
        'description': 'Your name',
    },
    'age': {
        'label': 'Age',
        'validators': [validators.NumberRange(min=14, max=99)],
    }
})
```

The class returned by `model_form()` can be used as a base class for forms mixing non-model fields and/or other model forms. For example:

```
# Generate a form based on the model.
BaseContactForm = model_form(Contact)

# Generate a form based on other model.
ExtraContactForm = model_form(MyOtherModel)

class ContactForm(BaseContactForm):
    # Add an extra, non-model related field.
    subscribe_to_news = f.BooleanField()

    # Add the other model form as a subform.
    extra = f.FormField(ExtraContactForm)
```

The class returned by `model_form()` can also extend an existing form class:

```
class BaseContactForm(Form):
    # Add an extra, non-model related field.
    subscribe_to_news = f.BooleanField()

# Generate a form based on the model.
ContactForm = model_form(Contact, base_class=BaseContactForm)
```

`wtforms_appengine.db.`**`model_form`**(*model*, *base_class=Form*, *only=None*, *exclude=None*, *field_args=None*, *converter=None*)

Creates and returns a dynamic `wtforms.Form` class for a given `db.Model` class. The form class can be used as it is or serve as a base for extended form classes, which can then mix non-model related fields, subforms with other model forms, among other possibilities.

Parameters

- **`model`** – The `db.Model` class to generate a form for.
- **`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.

1.2 Datastore-backed Fields

class `wtforms_appengine.fields.`**ReferencePropertyField**(*default field arguments*, *reference_class=None*, *get_label=None*, *allow_blank=False*, *blank_text=''*)

A field for `db.ReferenceProperty`. The list items are rendered in a select.

Parameters

- **reference_class** – A db.Model class which will be used to generate the default query to make the list of items. If this is not specified, The *query* property must be overridden before validation.
- **get_label** – If a string, use this attribute on the model class as the label associated with each option. 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.
- **allow_blank** – If set to true, a blank choice will be added to the top of the list to allow *None* to be chosen.
- **blank_text** – Use this to override the default blank option's label.

class `wtforms_appengine.fields.`**StringListPropertyField**(*default field arguments*)

A field for `db.StringListProperty`. The list items are rendered in a textarea.

class `wtforms_appengine.fields.`**IntegerListPropertyField**(*default field arguments*)

A field for `db.StringListProperty`. The list items are rendered in a textarea.

class `wtforms_appengine.fields.`**GeoPtPropertyField**(*default field arguments*)

1.3 NDB

WTForms now includes support for NDB models and can support mapping the relationship fields as well as generating forms from models.

class `wtforms_appengine.fields.`**KeyPropertyField**(*default field arguments*, *reference_class=None*, *get_label=None*, *allow_blank=False*, *blank_text=''*)

A field for `ndb.KeyProperty`. The list items are rendered in a select.

Parameters

- **reference_class** – A db.Model class which will be used to generate the default query to make the list of items. If this is not specified, The *query* property must be overridden before validation.

- **get_label** – If a string, use this attribute on the model class as the label associated with each option. 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.
- **allow_blank** – If set to true, a blank choice will be added to the top of the list to allow *None* to be chosen.
- **blank_text** – Use this to override the default blank option's label.

wtforms_appengine.ndb.**model_form**(*model*, *base_class=Form*, *only=None*, *exclude=None*, *field_args=None*, *converter=None*)

Creates and returns a dynamic `wtforms.Form` class for a given `ndb.Model` class. The form class can be used as it is or serve as a base for extended form classes, which can then mix non-model related fields, subforms with other model forms, among other possibilities.

Parameters

- **model** – The `ndb.Model` class to generate a form for.
- **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.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

W

G

I

K

M

R

S

W