django-etc Documentation

Release 1.2.0

Igor ‘idle sign‘ Starikov

Aug 05, 2020

Contents

https://github.com/idlesign/django-etc

CHAPTER 1

Description

Tiny stuff for Django that won't fit into separate apps.

Note: Add the **etc** application to INSTALLED_APPS in your settings file (usually 'settings.py').

CHAPTER 2

Requirements

1. Python 3.5+
2. Django 1.8+

CHAPTER 3

Table of Contents

3.1 Model Related Bits

3.1.1 InheritedModel

etc.toolbox.InheritedModel allows to override fields attributes in inherited models.

Mix in this class into target model (inherit from it) and define *Fields* class inside it to be able to customize field attributes (e.g. texts) of a base-parent model.

```
from etc.toolbox import InheritedModel


class MyAbstractModel(models.Model):

    code = models.CharField('dummy', max_length=64)
    expired = models.BooleanField('Expired', help_text='dummy')

    class Meta:
        abstract = True


class SecretModel(InheritedModel, MyParentModel):  # NOTE: InheritedModel must go␣
↪first.

    time_created = models.DateTimeField('Date created', auto_now_add=True)

    class Fields:  # Defining a class with fields custom fields data.
        code = 'Secret code'  # This is treated as verbose_name.
        expired = {'help_text': 'This code is expired.'}


class NonSecretModel(InheritedModel, MyParentModel):
```

(continues on next page)

(continued from previous page)

```
    code = models.CharField('dummy', max_length=128, unique=True, editable=False)

    class Fields:
        code = 'Non-secret code'
        expired = {'help_text': 'Do not check it. Do not.'}
```

3.1.2 Template Tags

model_meta

- **model_meta_verbose_name** tag.

  Returns model verbose name singular.

  ```
  {% load model_meta %}
  {% model_meta_verbose_name my_model %}
  ```

- **model_meta_verbose_name_plural** tag.

  Returns model verbose name plural.

  ```
  {% load model_meta %}
  {% model_meta_verbose_name_plural my_model %}
  ```

model_field

- **model_field_verbose_name** tag.

  Returns model field verbose name.

  ```
  {% load model_field %}
  {% model_field_verbose_name from my_model.fieldname %}
  ```

- **model_field_help_text** tag.

  Returns model field help text.

  ```
  {% load model_field %}
  {% model_field_help_text from my_model.fieldname %}
  ```

Both template tags are capable to redirect output into a template context variable using *as* clause. That could be useful if you have a set of homogeneous objects (e.g. QuerySet or Page) and want to get verbose name just once:

```
{% model_field_verbose_name from my_models_set.fieldname as title_fieldname %}
```

Note: *fieldname* could be a literal field name or a template variable containing the name.

3.1.3 Getting models

get_model_class_from_string

- **etc.toolbox.get_model_class_from_string** allows getting model class from its string representation.

Returns a certain model as defined in a string formatted <app_name>.<model_name>.

```
model = get_model_class_from_string('myapp.MyModel')
```

get_model_class_from_settings

etc.toolbox.get_model_class_from_settings allows getting model class from its string representation in settings module.

This might be handy if you allow users of your app to extend/override your built-in models:

```
myapp/settings.py

    from django.conf import settings

    # This allows users to set MYAPP_MY_MODEL in settings.py of their projects.
    MY_MODEL = getattr(settings, 'MYAPP_MY_MODEL', 'myapp.MyModel')


myapp/utils.py

    from myapp import settings

    def get_my_model():
        return get_model_class_from_settings(settings, 'MY_MODEL')
```

After that `get_my_model` will always return an appropriate model class object even if it is customized by a user.

3.1.4 Models choices

ChoicesEnumMixin

etc.toolbox.ChoicesEnumMixin helps to define choices for models using `Enum` from Python 3.

Could be used in conjunction with `get_choices` for convenience.

```
from enum import Enum, unique

@unique
class Role(ChoicesEnumMixin, Enum):

    # Define your Enum with mixin:
    # Item values could be tuples: (value, title, hint).

    APPLICANT = 0, 'Title', 'Hint'
    ADMIN = 1, 'Administrator'
    MEMBER = 2

class MyChoiceModel(models.Model):

    # Use the enum in field declaration.
    role = models.PositiveIntegerField(choices=get_choices(Role), default=Role.MEMBER)

# Filter objects by enum values.
members = MyChoiceModel.objects.filter(role=Role.MEMBER)
```

(continues on next page)

(continued from previous page)

```
# Access titles and hints registries
# (ordered dictionaries, indexed by values):
titles = Role.titles
hints = Role.hints
```

choices_list

etc.toolbox.choices_list helps to define choices for models, that could be addressed later as dictionaries.

To be used in conjunction with `get_choices`.

```
class MyModel(models.Model):

    TYPE_ONE = 1
    TYPE_TWO = 2

    TYPES = choices_list(
        (TYPE_ONE, 'Type one title'),
        (TYPE_TWO, 'Type two title'),
    )

    type = models.PositiveIntegerField('My type', choices=get_choices(TYPES),␣
↪default=TYPE_TWO)

    def get_display_type(self):
        return self.TYPES[self.type]
```

get_choices

etc.toolbox.get_choices returns model field choices from a given choices list.

Choices list is defined with `choices_list` or `ChoicesEnumMixin`, see above.

3.2 Forms Related Bits

3.2.1 set_form_widgets_attrs

etc.toolbox.set_form_widgets_attrs allows bulk apply HTML attributes to every field widget of a given form.

```
set_form_widgets_attrs(my_form, {'class': 'clickable'})
```

3.3 Various utils

3.3.1 import_app_module

etc.toolbox.import_app_module imports and returns a module from a specific app by its name.

If your application provides some kind of tooling for others and you know that configuration for this tooling could be found in a certain module within a thirdparty app you can use this function to load such a module by its name.

```
from etc.toolbox import import_app_module


module = import_app_module('someapp', 'mymodule')  # Get `mymodule` module from␣
↪`someapp` application.
```

3.3.2 import_project_modules

etc.toolbox.import_project_modules imports modules from registered apps using given module name and returns them as a list.

This is an automation for *import_app_module()* described above to load all modules from every app in a project.

```
from etc.toolbox import import_project_modules


all_modules = import_project_modules('mymodule')  # Get `mymodule` module from every␣
↪app in a project.
```

3.3.3 get_site_url

etc.toolbox.get_site_url does its best to provide you with a site URL where request object is unavailable.

On occasions when you do not have a request object to get current site URL from (e.g. background tasks) this function tries to get it from *environment* and *settings*, using the following order:

1. (SITE_PROTO or SITE_SCHEME) + SITE_DOMAIN
2. SITE_URL
3. Django Sites contrib
4. Request object (if available)

```
from etc.toolbox import get_site_url


my_url = get_site_url()
```

3.3.4 *etc_misc* Template Tags

- **site_url** tag.

  Does its best to provide you with a site URL whether request object is unavailable or not. See `get_site_url` description above.

  ```
  {% load etc_misc %}
  {% site_url %}
  ```

- **include_** tag.

  Similar to built-in `include` template tag, but allowing template variables to be used in template name and a fallback template, thus making the tag more dynamic.

> **Warning:** Requires Django 1.8+

```
{% load etc_misc %}
{% include_ "sub_{{ postfix_var }}.html" fallback "default.html" %}
```

3.4 Thirdparty Related Bits

3.4.1 *gravatar* Template Tags

- **gravatar_get_url** tag.

  Returns Gravatar image URL for a given string or UserModel.

  Accepts `size` integer and `default` image identifier as a string

  (see http://ru.gravatar.com/site/implement/images/#default-image).

  ```
  {% load gravatar %}
  {% gravatar_get_url user_model %}
  ```

- **gravatar_get_img** tag.

  Returns Gravatar image HTML tag for a given string or UserModel.

  Accepts `size` integer and `default` image identifier as a string

  (see http://ru.gravatar.com/site/implement/images/#default-image).

  ```
  {% load gravatar %}
  {% gravatar_get_img user_model %}
  ```

CHAPTER 4

Get involved into django-etc

Submit issues. If you spotted something weird in application behavior or want to propose a feature you can do that at https://github.com/idlesign/django-etc/issues

Write code. If you are eager to participate in application development, fork it at https://github.com/idlesign/django-etc, write your code, whether it should be a bugfix or a feature implementation, and make a pull request right from the forked project page.

Spread the word. If you have some tips and tricks or any other words in mind that you think might be of interest for the others — publish it.