django-translated-fields Documentation

Release 0.12

Feinheit AG

Apr 08, 2022

Contents

Django model translation without magic-inflicted pain.

CHAPTER 1

Installation and usage

After installing `django-translated-fields` in your Python environment all you have to do is define `LANGUAGES` in your settings and add translated fields to your models:

```
from django.db import models
from django.utils.translation import gettext_lazy as _

from translated_fields import TranslatedField

class Question(models.Model):
    question = TranslatedField(
        models.CharField(_("question"), max_length=200),
    )
    answer = TranslatedField(
        models.CharField(_("answer"), max_length=200),
    )

    def __str__(self):
        return self.question
```

CHAPTER 2

Basic usage

Model fields are automatically created from the field passed to `TranslatedField`, one field per language. For example, with `LANGUAGES = [("en", "English"), ("de", "German"), ("fr", "French")]`, the following list of fields would be created: `question_en`, `question_de`, `question_fr`, `answer_en`, `answer_de`, and `answer_fr`.

This implies that when changing `LANGUAGES` you'll have to run `makemigrations` and `migrate` too.

No `question` or `answer` model field is actually created. The `TranslatedField` instance is a descriptor which by default acts as a property for the current language's field:

```
from django.utils.translation import override

question = Question(
    question_en="How are you?",
    question_de="Wie geht es Dir?",
    question_fr="Ça va?",
)

# The default getter automatically returns the value
# in the current language:
with override("en"):
    assert question.question == "How are you?"

with override("de"):
    assert question.question == "Wie geht es Dir?"

# The default setter can also be used to set the value
# in the current language:
with override("fr"):
    question.question = "Comment vas-tu?"

assert question.question_fr == "Comment vas-tu?"
```

`TranslatedField` has a `fields` attribute that returns a list of all the language fields created.

```
assert Question.answer.fields == ["answer_en", "answer_de", "answer_fr"]
```

For more attributes look at the *‘‘TranslatedField‘‘ instance API* section below.

`question` and `answer` can only be used with model instances, they do not exist in the database. If you want to use queryset methods which reference individual translated fields you have to use language-specific field names yourself. If you wanted to fetch only the english question and answer fields you could do this as follows:

```
questions = Question.objects.values_list("question_en", "answer_en")
```

Or better yet, using the `to_attribute` helper which automatically uses the active language (if you don't pass a specific language code as its second argument):

```
from django.utils.translation import override
from translated_fields import to_attribute

with override("en"):
    questions = Question.objects.values_list(
        to_attribute("question"), to_attribute("answer")
    )
```

CHAPTER 3

Changing field attributes per language

It is sometimes useful to have slightly differing model fields per language, e.g. for making the primary language mandatory. This can be achieved by passing a dictionary with keyword arguments per language as the second positional argument to `TranslatedField`.

For example, if you add a language to `LANGUAGES` when a site is already running, it might be useful to make the new language non-mandatory to simplify editing already existing data through Django's administration interface.

The following example adds `blank=True` to the spanish field:

```
from translated_fields import TranslatedField

class Question(models.Model):
    question = TranslatedField(
        models.CharField(_("question"), max_length=200),
        {"es": {"blank": True}},
    )
```

CHAPTER 4

Overriding attribute access (defaults, fallbacks)

There are no default values or fallbacks, only a wrapped attribute access. The default attribute getter and setter functions simply return or set the field for the current language (as returned by `django.utils.translation.get_language`). The default getter falls back to the first language of the field in case `get_language()` returns `None`. Apart from that the default getter has no safetyfeatures and may raise an `AttributeError` and the setter might set an attribute on the model instance not related to a model field.

Both getters and setters can be overridden by specifying your own `attrgetter` and `attrsetter` functions. E.g. you may want to specify a fallback to the default language (and at the same time allow leaving other languages' fields empty):

```
from django.conf import settings
from translated_fields import TranslatedField, to_attribute

def fallback_to_default(name, field):
    def getter(self):
        return getattr(
            self,
            to_attribute(name),
        ) or getattr(
            self,
            # First language acts as fallback:
            to_attribute(name, settings.LANGUAGES[0][0]),
        )
    return getter

class Question(models.Model):
    question = TranslatedField(
        models.CharField(_("question"), max_length=200, blank=True),
        {settings.LANGUAGES[0][0]: {"blank": False}},
        attrgetter=fallback_to_default,
    )
```

Maybe you're using locales with region codes such as `fr-fr` where you want to fall back to the language without a region code. An example `attrgetter` implementation follows:

```
from translated_fields import to_attribute

def fallback_to_all_regions(name, field):
    def getter(self):
        value = getattr(self, to_attribute(name), None)
        if value:
            return value
        return getattr(self, to_attribute(name, get_language().split("-")[0]))

    return getter
```

A custom `attrsetter` which always sets all fields follows (probably not very useful, but hopefully instructive):

```
def set_all_fields(name, field):
    def setter(self, value):
        for field in field.fields:
            setattr(self, field, value)
    return setter
```

CHAPTER 5

`TranslatedField` instance API

The `TranslatedField` descriptor has a few useful attributes (sticking with the model and field from the examples above):

- `Question.question.fields` contains the names of all automatically generated fields, e.g. `["question_en", "question_...", ...]`.
- `Question.question.languages` is the list of language codes.
- `Question.question.short_description` is set to the `verbose_name` of the base field, so that the translatable attribute can be nicely used e.g. in `ModelAdmin.list_display`.

CHAPTER 6

Using a different set of languages

It is also possible to override the list of language codes used, for example if you want to translate a sub- or superset of `settings.LANGUAGES`. Combined with `attrgetter` and `attrsetter` there is nothing stopping you from using this field for a different kind of translations, not necessarily bound to `django.utils.translation` or even languages at all.

CHAPTER 7

Translated attributes without model field creation

If model field creation is not desired, you may also use the `translated_attributes` class decorator. This only creates the attribute getter property:

```
from translated_fields import translated_attributes

@translated_attributes("attribute", "anything", ...)
class Test(object):
    attribute_en = "some value"
    attribute_de = "some other value"
```

CHAPTER 8

Model admin support

The `TranslatedFieldAdmin` class adds the respective language to the label of individual fields. Instead of three fields named "Question" you'll get the fields "Question [en]", "Question [de]" and "Question [fr]". It intentionally offers no functionality except for modifying the label of fields:

```
from django.contrib import admin
from translated_fields import TranslatedFieldAdmin
from .models import Question

@admin.register(Question)
class QuestionAdmin(TranslatedFieldAdmin, admin.ModelAdmin):
    pass

# For inlines:
# class SomeInline(TranslatedFieldAdmin, admin.StackedInline):
#     ...
```

As mentioned above, the `fields` attribute on the `TranslatedField` instance contains the list of generated fields. This may be useful if you want to customize various aspects of the `ModelAdmin` subclass. An example showing various techniques follows:

```
from django.contrib import admin
from django.utils.translation import gettext_lazy as _
from translated_fields import TranslatedFieldAdmin, to_attribute
from .models import Question

@admin.register(Question)
class QuestionAdmin(TranslatedFieldAdmin, admin.ModelAdmin):
    # Pack question and answer fields into their own fieldsets:
    fieldsets = [
        (_("question"), {"fields": Question.question.fields}),
        (_("answer"), {"fields": Question.answer.fields}),
    ]

    # Show all fields in the changelist:
```

(continues on next page)

(continued from previous page)

```
    list_display = [
        *Question.question.fields,
        *Question.answer.fields
    ]

    # Order by current language's question field:
    def get_ordering(self, request):
        return [to_attribute("question")]
```

Note: It's strongly recommended to set the `verbose_name` of fields when using `TranslatedFieldAdmin`, the first argument of most model fields. Otherwise, you'll get duplicated languages, e.g. "Question en [en]".

CHAPTER 9

Forms

django-translated-fields provides a helper when you want form fields' labels to contain the language code. If this sounds useful to you do this:

```
from django import forms
from translated_fields.utils import language_code_formfield_callback
from .models import Question

class QuestionForm(forms.ModelForm):
    formfield_callback = language_code_formfield_callback

    class Meta:
        model = Question
        fields = [
            *Question.question.fields,
            *Question.answer.fields
        ]
```

You may also globally configure language code labels to be shown within a block:

```
from translated_fields import show_language_code

def view(request):
    form = ...
    with show_language_code(True):
        return render(request, "...", {"form": form})
```

Please note that the response has to be rendered within the `show_language_code` block. This doesn't happen automatically when using Django's `TemplateResponse` objects.

CHAPTER 10

Other features

There is no support for automatically referencing the current language's field in queries or automatically adding fields to admin fieldsets and whatnot. The code required for these features isn't too hard to write, but it is hard to maintain down the road which contradicts my goal of writing low maintenance software. Still, feedback and pull requests are very welcome! Please run the style checks and test suite locally before submitting a pull request though – all that this requires is running tox.

CHAPTER 11

Change log

11.1 Next version

11.2 0.12 (2022-04-08)

- **BACKWARDS INCOMPATIBLE:** Made the `field` keyword argument to the attrgetter and attrsetter functions mandatory. django-translated-fields raised a deprecation warning if an attrgetter or attrsetter didn't support it since 0.8 (released in 2019) so this shouldn't be a problem for anyone, hopefully.
- Made `language_code_formfield_callback` preserve the lazyness of the underlying `verbose_name`.
- Stopped overwriting language-specific `verbose_name` values.
- Made `translated_attributes` fall back to the first entry in `settings.LANGUAGES` when no language is active. This previously just crashed with an `AttributeError` (but not caused by non-existant attributes on the model instance, but caused by the fact that the getter didn't receive a `TranslatedField` instance)
- Added Python 3.10, Django 4.0 to the CI.
- Dropped Python < 3.8, Django < 3.2.
- Added pre-commit.

11.3 0.11 (2021-04-12)

- Changed `TranslatedFieldAdmin` to not blindly call `.render()` on all responses, just on those actually having such an attribute.
- Changed `fallback_to_default`, `fallback_to_any` and `TranslatedFieldWithFallback` to not fail with an attribute error if no language is active at all.
- Renamed the main branch of the repository to `main`.

- Switched from Travis CI to GitHub actions.
- Verified compatibility with Python 3.9 and Django 3.2.
- Renamed the main branch to `main`.
- Switched to a declarative setup (`setup.py` and `setup.cfg`).
- Fixed a bug where field ordering was incorrect when overriding the `languages` list of a translated field with a list longer than `settings.LANGUAGES`.

11.4 0.10 (2020-07-27)

- Changed the `verbose_name` of fields generated by `TranslatedField` to return the language_code when inside a `with translated_fields.show_language_code(True):` block. This change introduces a dependency on contextvars which is automatically installed from PyPI in Python<3.7.
- Completely overhauled `TranslatedFieldAdmin` to take advantage of `show_language_code`, making it possible to use translated fields together with `list_display_links`, `list_editable`, `readonly_fields` etc.
- Dropped compatibility guarantees for Python 3.5.
- Stopped shadowing import errors.

11.5 0.9 (2020-05-14)

- Changed `fallback_to_any` to also accept the `field` as an argument, which avoids warnings.
- Added Django 3.1 to the matrix, dropped unmaintained Django versions (all versions < 2.2).
- Fixed a compatibility problem with Django 3.1 by importing `FieldDoesNotExist` from `django.core.exceptions`.

11.6 0.8 (2019-06-26)

- Added an optional `field` argument to the `attrgetter` and `attrsetter` functions.
- Added a utils module intended to contain common applications of translated fields. For now, `TranslatedFieldWithFallback` creates a field where all languages but the primary language (the first language in `settings.LANGUAGES` resp. the first entry in the `languages` argument if given) are optional and and fall back to the field in the primary language if their value is falsy.
- Added a `fallback_to_any` translated attribute getter which returns either the attribute in the current language or in any of the languages.
- `fallback_to_default` and by extension `TranslatedFieldWithFallback` no longer fall back to the first entry in `settings.LANGUAGES` but to the fields' first language (which is the same except when overriding the `languages` list in the `TranslatedField` instantiation).
- Added a `field` keyword argument to the attrgetter and attrsetter calls. If an existing custom getter or setter does not support the argument you'll get a deprecation warning.

11.7 0.7 (2018-10-17)

- Reused Django's own machinery for displaying data in the changelist instead of playing catch-up ourselves.
- Moved the `list_display_column` helper functionality into the `TranslatedFieldAdmin` class and made its application automatic as long as you're not overriding `get_list_display`.

11.8 0.6 (2018-10-17)

- Added an example and an explanation how to best customize the administration interface when using django-translated-fields.
- Added Django 2.1 to the Travis CI test matrix (no changes were necessary for compatibility).
- Made pull requests not following the black coding style fail.
- Added the "production/stable" development status trove identifier.
- Dropped Python 3.4 from the build matrix.
- Added a `list_display_column` helper for showing language codes in column titles.

11.9 0.5 (2018-06-14)

- Replaced the `verbose_name_with_language` option and the `verbose_name` mangling it does with `TranslatedFieldAdmin` which offers the same functionality, but restricted to the admin interface.

11.10 0.4 (2018-06-14)

- Switched the preferred quote to `"` and started using black to automatically format Python code.
- Added Python 3.4 to the test matrix.
- Made documentation better.

11.11 0.3 (2018-05-03)

- Added documentation.
- Converted the `TranslatedField` into a descriptor, and made available a few useful fields on the descriptor instance.
- Made it possible to set the value of the current language's field, and added another keyword argument for replacing the default `attrsetter`.
- Made `to_attribute` fall back to the current language.
- Added exports for `to_attribute`, `translated_attrgetter` and `translated_attrsetter` to `translated_fields`.
- Added an `attrgetter` argument to `translated_attributes`.

11.12 0.2 (2018-04-30)

- By default the language is appended to the `verbose_name` of fields created by `TranslatedField`. Added the `verbose_name_with_language=True` parameter to `TranslatedField` which allows skipping this behavior.
- Added a `languages` keyword argument to `TranslatedField` to allow specifying a different set of language-specific fields than the default of the `settings.LANGUAGES` setting.
- Added a `attrgetter` keyword argument to `TranslatedField` to replace the default implementation of language-specific attribute getting.
- Added the possibility to override field keyword arguments for specific languages, e.g. to only make a single language field mandatory and implement your own fallback via `attrgetter`.

11.13 0.1 (2018-04-18)

- Initial release!