Django Autoslug Documentation

Release 1.9

Andy Mikhailenko

Jun 14, 2023

Contents

Django-autoslug is a reusable Django library that provides an improved slug field which can automatically:

a) populate itself from another field,

b) preserve uniqueness of the value and

c) use custom `slugify()` functions for better i18n.

The field is highly configurable.

CHAPTER 1

Requirements

Python 3.7+ or PyPy.

Django 3.2 or higher.

It may be possible to successfully use django-autoslug in other environments but they are not tested.

Note: PyPy3 is not officially supported only because there were some problems with permissions and `__pycache__` on CI unrelated to django-autoslug itself.

CHAPTER 2

Installation

```
python -m pip install django-autoslug
```

CHAPTER 3

Examples

A simple example:

```
from django.db.models import CharField, Model
from autoslug import AutoSlugField

class Article(Model):
    title = CharField(max_length=200)
    slug = AutoSlugField(populate_from='title')
```

More complex example:

```
from django.db.models import CharField, DateField, ForeignKey, Model
from django.contrib.auth.models import User
from autoslug import AutoSlugField

class Article(Model):
    title = CharField(max_length=200)
    pub_date = DateField(auto_now_add=True)
    author = ForeignKey(User)
    slug = AutoSlugField(populate_from=lambda instance: instance.title,
                         unique_with=['author__name', 'pub_date__month'],
                         slugify=lambda value: value.replace(' ','-'))
```

CHAPTER 4

Documentation

See the complete documentation on ReadTheDocs. It is built automatically for the latest version.

CHAPTER 5

Community

This application is maintained by Justin Mayer. It was initially created by Andy Mikhailenko and then improved by other developers. They are listed in `AUTHORS.rst`.

Please feel free to file issues and/or submit patches.

See `CONTRIBUTING.rst` for hints related to the preferred workflow.

CHAPTER 6

Licensing

Django-autoslug is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

Django-autoslug is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this program; see the file COPYING.LESSER. If not, see GNU licenses.

See detailed documentation with real-world examples:

6.1 Fields

class autoslug.fields.**AutoSlugField**(**args*, ***kwargs*)

AutoSlugField is an extended SlugField able to automatically resolve name clashes.

AutoSlugField can also perform the following tasks on save:

- populate itself from another field (using *populate_from*),
- use custom *slugify* function (using *slugify* or *Settings*), and
- preserve uniqueness of the value (using *unique* or *unique_with*).

None of the tasks is mandatory, i.e. you can have auto-populated non-unique fields, manually entered unique ones (absolutely unique or within a given date) or both.

Uniqueness is preserved by checking if the slug is unique with given constraints (*unique_with*) or globally (*unique*) and adding a number to the slug to make it unique.

Parameters

- **always_update** – boolean: if True, the slug is updated each time the model instance is saved. Use with care because cool URIs don't change (and the slug is usually a part of

object's URI). Note that even if the field is editable, any manual changes will be lost when this option is activated.

- **populate_from** – string or callable: if string is given, it is considered as the name of attribute from which to fill the slug. If callable is given, it should accept *instance* parameter and return a value to fill the slug with.
- **sep** – string: if defined, overrides default separator for automatically incremented slug index (i.e. the "-" in "foo-2").
- **slugify** – callable: if defined, overrides *AUTOSLUG_SLUGIFY_FUNCTION* defined in *Settings*.
- **unique** – boolean: ensure total slug uniqueness (unless more precise *unique_with* is defined).
- **unique_with** – string or tuple of strings: name or names of attributes to check for "partial uniqueness", i.e. there will not be two objects with identical slugs if these objects share the same values of given attributes. For instance, `unique_with='pub_date'` tells AutoSlugField to enforce slug uniqueness of all items published on given date. The slug, however, may reappear on another date. If more than one field is given, e.g. `unique_with=('pub_date', 'author')`, then the same slug may reappear within a day or within some author's articles but never within a day for the same author. Foreign keys are also supported, i.e. not only *unique_with='author'* will do, but also *unique_with='author__name'*.

Note: always place any slug attribute *after* attributes referenced by it (i.e. those from which you wish to *populate_from* or check *unique_with*). The reasoning is that autosaved dates and other such fields must be already processed before using them in the AutoSlugField.

Example usage:

```
from django.db import models
from autoslug import AutoSlugField

class Article(models.Model):
    '''An article with title, date and slug. The slug is not totally
    unique but there will be no two articles with the same slug within
    any month.
    '''
    title = models.CharField(max_length=200)
    pub_date = models.DateField(auto_now_add=True)
    slug = AutoSlugField(populate_from='title', unique_with='pub_date__month')
```

More options:

```
# slugify but allow non-unique slugs
slug = AutoSlugField()

# globally unique, silently fix on conflict ("foo" --> "foo-1".."foo-n")
slug = AutoSlugField(unique=True)

# autoslugify value from attribute named "title"; editable defaults to False
slug = AutoSlugField(populate_from='title')

# same as above but force editable=True
slug = AutoSlugField(populate_from='title', editable=True)
```

(continues on next page)

(continued from previous page)

```
# ensure that slug is unique with given date (not globally)
slug = AutoSlugField(unique_with='pub_date')

# ensure that slug is unique with given date AND category
slug = AutoSlugField(unique_with=('pub_date','category'))

# ensure that slug is unique with an external object
# assuming that author=ForeignKey(Author)
slug = AutoSlugField(unique_with='author')

# ensure that slug is unique with a subset of external objects (by lookups)
# assuming that author=ForeignKey(Author)
slug = AutoSlugField(unique_with='author__name')

# mix above-mentioned behaviour bits
slug = AutoSlugField(populate_from='title', unique_with='pub_date')

# minimum date granularity is shifted from day to month
slug = AutoSlugField(populate_from='title', unique_with='pub_date__month')

# autoslugify value from a dynamic attribute (i.e. a method)
slug = AutoSlugField(populate_from='get_full_name')

# autoslugify value from a custom callable
# (ex. usage: user profile models)
slug = AutoSlugField(populate_from=lambda instance: instance.user.get_full_name())

# specify model manager for looking up slugs shared by subclasses

class Article(models.Model):
    '''An article with title, date and slug. The slug is not totally
    unique but there will be no two articles with the same slug within
    any month.
    '''
    objects = models.Manager()
    title = models.CharField(max_length=200)
    slug = AutoSlugField(populate_from='title', unique_with='pub_date__month',␣
↪manager=objects)

class NewsArticle(Article):
    pass

# autoslugify value using custom `slugify` function
from autoslug.settings import slugify as default_slugify
def custom_slugify(value):
    return default_slugify(value).replace('-', '_')
slug = AutoSlugField(slugify=custom_slugify)
```

deconstruct()

Return enough information to recreate the field as a 4-tuple:

- The name of the field on the model, if contribute_to_class() has been run.
- The import path of the field, including the class:e.g. django.db.models.IntegerField This should be the most portable version, so less specific may be better.
- A list of positional arguments.

- A dict of keyword arguments.

Note that the positional or keyword arguments must contain values of the following types (including inner values of collection types):

- None, bool, str, int, float, complex, set, frozenset, list, tuple, dict
- UUID
- datetime.datetime (naive), datetime.date
- top-level classes, top-level functions - will be referenced by their full import path
- Storage instances - these have their own deconstruct() method

This is because the values here must be serialized into a text format (possibly new Python code, possibly JSON) and these are the only types with encoding handlers defined.

There's no need to return the exact way the field was instantiated this time, just ensure that the resulting field is the same - prefer keyword arguments over positional ones, and omit parameters with their default values.

pre_save(*instance*, *add*)
Return field's value just before saving.

6.2 Settings

Django settings that affect django-autoslug:

AUTOSLUG_SLUGIFY_FUNCTION Allows to define a custom slugifying function.

The function can be repsesented as string or callable, e.g.:

```
# custom function, path as string:
AUTOSLUG_SLUGIFY_FUNCTION = 'some_app.slugify_func'

# custom function, callable:
AUTOSLUG_SLUGIFY_FUNCTION = some_app.slugify_func

# custom function, defined inline:
AUTOSLUG_SLUGIFY_FUNCTION = lambda slug: 'can i haz %s?' % slug
```

If no value is given, default value is used.

Default value is one of these depending on availability in given order:

- *unidecode.unidecode()* if Unidecode is available;
- *pytils.translit.slugify()* if pytils is available;
- *django.template.defaultfilters.slugify()* bundled with Django.

django-autoslug also ships a couple of slugify functions that use the translitcodec Python library, e.g.:

```
# using as many characters as needed to make a natural replacement
AUTOSLUG_SLUGIFY_FUNCTION = 'autoslug.utils.translit_long'

# using the minimum number of characters to make a replacement
AUTOSLUG_SLUGIFY_FUNCTION = 'autoslug.utils.translit_short'

# only performing single character replacements
AUTOSLUG_SLUGIFY_FUNCTION = 'autoslug.utils.translit_one'
```

AUTOSLUG_MODELTRANSLATION_ENABLE Django-autoslug support of modeltranslation is still experimental. If you wish to enable it, please set this option to *True* in your project settings. Default is *False*.

6.3 Authors

The django-autoslug library is currently maintained by:

- Justin Mayer <https://justinmayer.com/>

It was originally created by:

- Andy Mikhailenko <neithere@gmail.com>

Here is a probably incomplete list of contributors – people who have submitted patches, reported bugs, added translations and generally made django-autoslug better:

- Steve Steiner
- Blake Imsland
- Ollie Rutherfurd
- Mikhail Korobov
- Remco Wendt
- Johan Charpentier
- Nicolás Echániz
- Aaron VanDerlip
- Thomas Woolford
- Jannis Leidel
- Caio Ariede
- Venelin Stoykov
- Bertrand Bordage
- Davor Teskera
- Florian Apolloner
- Fabio Caccamo
- Thomas Schreiber
- Mike Urbanski
- Vadim Iskuchekov
- kane-c
- Julien Dubiel
- Tony Shtarev
- Éloi Rivard
- Peter Baumgartner
- Jernej Kos
- Sutrisno Efendi

- Your Name Here ;)

6.4 Changelog

6.4.1 1.9.9 - 2023-04-03

- Prevent situation in which slug could end in dash or underscore
- Remove support for end-of-life Python & Django versions

6.4.2 1.9.8 - 2020-07-22

Move FieldDoesNotExist import for compatibility with Django 3.1

6.4.3 1.9.7 - 2020-04-14

Fix assertion error on empty slug

6.4.4 1.9.6 - 2019-07-30

Handle timezones for datetime fields

6.4.5 1.9.5 - 2019-07-28

Add license to sdist and wheels

6.4.6 Version 1.9.4

New features:

- Add *manager_name* kwarg to enable using custom managers from abstract models
- Add compatibility for Django versions 1.10, 1.11, 2.0, and 2.1
- Transfer project to new maintainer

6.4.7 Version 1.9.3

- Add allow_unicode attribute for django 1.9 compatibility.
- Tweak packaging

6.4.8 Version 1.9.1, 1.9.2

Bugs fixed:

- #43 — Packaging error

6.4.9 Version 1.9.0

Backwards incompatible changes:

- Limited supported versions of Python to 2.7, 3.5 and PyPy.
- Limited supported Django versions to 1.7.10 and higher.
- Turned off modeltranslation support by default (can be enabled)

Bugs fixed:

- #25 — max_length ignored in django 1.7 migrations.
- #42 — Added setting to enable/disable modeltranslation support.

Other changes:

- Converted the test suite from doctest to unittest.
- The project has moved from Bitbucket to GitHub.

6.4.10 Old versions

Changelog before extracting to a separate repository:

```
changeset:   23:34210c5b5b72
user:        Andy Mikhailenko <neithere@gmail.com>
date:        Sat Sep 27 05:55:42 2008 +0600
summary:     Fixed bug in AutoSlugField: uniqueness check by date was broken

changeset:   22:8b13c99f2164
user:        Andy Mikhailenko <neithere@gmail.com>
date:        Sat Sep 27 04:14:04 2008 +0600
summary:     Rewrite AutoSlugField. Add optional attributes "unique" and "unique_for_
↪date". Preserve "populate_from" as optional.

changeset:   21:07aa85898221
parent:      19:ae6294ba1162
user:        Andy Mikhailenko <neithere@gmail.com>
date:        Fri Sep 26 23:57:29 2008 +0600
summary:     Use pytils for transliteration is AutoSlugField

changeset:   12:e8b861b632d7
user:        Andy Mikhailenko <neithere@gmail.com>
date:        Wed Aug 06 07:26:39 2008 +0600
summary:     Fix bug in custom_forms.auto_slug_field (missing import directive)

changeset:   10:ac217f7edb53
user:        Andy Mikhailenko <neithere@gmail.com>
date:        Wed Aug 06 07:19:17 2008 +0600
summary:     Add custom_models, including AutoSlugField
```

6.5 Indices and tables

- genindex
- modindex

- search

Python Module Index

a

Index