django-smartfields Documentation

Release 1.1.3

Alexey Kuleshevich

Nov 27, 2020

Contents

Django Model Fields that are smart

This application introduces a totally new way of handling field's values through unique ways they are assigned and processed. It is so simple that nothing needs to be done in order to start using it, yet it is so powerful, that it can handle automatic image and video file conversions with a simple specification of a conversion function. Check it out, and it will forever change the way you handle Model Fields.

CHAPTER 1

Installation

```
pip install django-smartfields
```

CHAPTER 2

Latest build

Forkme on Github: django-smartfields

CHAPTER 3

Introduction

Here is a short introduction of how this app works and a simple example how it can be used.

First of all, as name suggests, it mainly deals with Model Fields, hence it is supplied with a custom version of every Django's Field. There is no difference form original versions of fields in terms of interaction with database, forms or with any other Django codebase, so both kinds of fields can be used together safely and interchangeably. Main distinction form Django's fields is that all smartfields accept a keyword argument `dependencies`, which should be a list of *`Dependency's`* or *`FileDependency's`*.

Dependency is a concept that allows you to change the value of any field or an attribute attached to the model instance, including the field *Dependency* which it is specified for. Each *Dependency* handles the value from a field through *Processors* which are functions that can be accepted as `default`, `pre_processor` and `processor` kwargs. An actual model attribute or a field which a processed value will be assigned to is specified by one or none of the kwargs `suffix` and `attname`. More details on those see documentation in *Dependencies* and *Processors* sections, but for now let's see a couple of simple examples.

3.1 Example

Let's say we have a Product model where a slug needs to be automatically generated from product's name and also properly modified to look like a slug.

```
from django.db.models import models
from django.utils.text import slugify
from smartfields import fields
from smartfields.dependencies import Dependency

def name_getter(value, instance, **kwargs):
    return instance.name

class Product(models.Model):
    name = models.CharField(max_length=255)
    slug = fields.SlugField(dependencies=[
```

(continues on next page)

(continued from previous page)

```
        Dependency(default=name_getter, processor=slugify)
    ])
```

Here is what will happen in above example whenever an instance of `Product` is created:

- Whenever `Product` is initilized and `slug` field is empty, it will attempt to get a value from `name` field. In case when it is still empty before model is being saved it will attempt to get the value again, all because of `default` function `name_getter`.
- Right before the model is saved `processor` function `slugify` will be invoked, and value of the field from `name` will be modified to look like a slug. Important part is, processor will be invoked only whenever the value of `slug` field has changed.

CHAPTER 4

Important Perculiarities

- Fields are processed in order they are specified in a Model.
- Dependencies are processed in the order they are speciefied in the `dependencies` list, except the ones with `async` flag, these are processed last, but also in the order they were specified.

CHAPTER 5

More Details

5.1 Dependencies

class smartfields.dependencies.**Dependency**

__init__(*attname=None*, *suffix=None*, *processor=None*, *pre_processor=None*, *async=False*, *default=NOT_PROVIDED*, *processor_params=None*, *uid=None*)

Parameters

- **attname** (*str*) – Name of an attribute or an existing field that dependecy will assign a value to. Cannot be used together with **suffix**.
- **suffix** (*str*) – Will be used together with a field name in generating an **attname** in format *field_name_suffix*. Generated name can refer to an attribute or an existing field that dependecy will assign a value to. Cannot be used together with **attname**.
- **processor** – A function that takes field's value as an argument or an instance of a class derived from *BaseProcessor*. In a latter case it will receive all arguments: `value`, `instance`, `field`, `field_value`, `dependee`, `stashed_value` plus any custom kwargs. If a class is passed instead of it's instance it will be instantiated, to prevent a common mistake.
- **pre_processor** –
- **async** –
- **default** –
- **processor_params** –
- **uid** –

class smartfields.dependencies.**FileDependency**

Because FileFields are handled in a different way then regular fields we need a different type of dependecy too.

__init__(*upload_to=”, storage=None, keep_orphans=KEEP_ORPHANS, **kwargs*)

keyword upload_to

keyword storage

keyword keep_orphans

5.2 Processors

class smartfields.processors.**BaseProcessor**

__init__(***kwargs*)

process(*value, instance=None, field=None, dependee=None, stashed_value=None, **kwargs*)

Parameters

- **value** – New value that is being assigned to the parent field.
- **instance** – Model instance that a field is attached to.
- **field** – Parent field instance.
- **dependee** – Instance of a field that depends on the `field`. It is decided by the `attname` or `suffix` argument to the
- **stashed_value** – This is a previous value that a `dependee` field was holding. Very useful for comparing it to new values.

class smartfields.processors.**BaseFileProcessor**

get_ext(*format=None, **kwargs*)

class smartfields.processors.**RenameFileProcessor**

class smartfields.processors.**ExternalFileProcessor**

class smartfields.processors.**FFMPEGProcessor**

__init__()

process(*value, **kwargs*)

Here is an example of how to convert a video to MP4 format. In this example every time `MediaModel` is instantiated *`FileDependency`* will automatically attach another field like attribute to the model `video_mp4`. Moreover, whenever a new video file is uploaded or simply assigned to a `video` field, it will use *`FFMPEGProcessor`* and `ffmpeg` to convert that video file to mp4 format and will assign it the same name as original video with `mp4` suffix and file extension. While converting a video file it will set progress between 0.0 and 1.0, which can be retrieved from field’s status.

```
from django.db import models
from smartfields import fields, dependencies
from smartfields.processors import FFMPEGProcessor

class MediaModel(models.Model):
    video = fields.FileField(dependencies=[
```

(continues on next page)

(continued from previous page)

```
        dependencies.FileDependency(suffix='mp4', processor=FFMPEGProcessor(
            vbitrate = '1M',
            maxrate = '1M',
            bufsize = '2M',
            width = 'trunc(oh*a/2)*2', # http://ffmpeg.org/ffmpeg-all.html#scale
            height = 720,
            threads = 0, # use all cores
            abitrate = '96k',
            format = 'mp4',
            vcodec = 'libx264',
            acodec = 'libfdk_aac'))])
```

class smartfields.processors.**CloudFFMEGPRocessor**

__init__()

process(*value*, ***kwargs*)

Here is an example of how to upload file in custom storage use *django-storages*. Each storage backend has its own unique settings you will need to add to your *settings.py* file.

```
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
STATICFILES_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
```

```
from django.db import models
from smartfields import fields, dependencies
from smartfields.processors import CloudFFMEGPRocessor

class MediaModel(models.Model):
    video = fields.FileField(dependencies=[
        dependencies.FileDependency(suffix='mp4', processor=CloudFFMEGPRocessor(
            vbitrate = '1M',
            maxrate = '1M',
            bufsize = '2M',
            width = 'trunc(oh*a/2)*2', # http://ffmpeg.org/ffmpeg-all.html#scale
            height = 720,
            threads = 0, # use all cores
            abitrate = '96k',
            format = 'mp4',
            vcodec = 'libx264',
            acodec = 'libfdk_aac'))])
```

CHAPTER 6

Project Info

6.1 Changelog

6.1.1 1.1.3

- Addition of *CloudImageProcessor* and *CloudFFMEGPRocessor*

6.1.2 1.1.2

- Support for Django=3.1.*

6.1.3 1.1.1

- Support for Django=3.0.*

6.1.4 1.1.0

- renamed `Dependency.async` to `Dependency.async_`. Fix for #16. Thanks @zglennie
- Fix compatibility with `Django=2.x`:
  - Added `app_name='smartifelds'` to `urls.py` file
  - Stop using `_size` and `_set_size()` attributes in `NamedTemporaryFile`, since those where only available in `Django=1.x`

6.1.5 1.0.7

- added `gis` fields.
- made `lxml` a default parser for HTMLProcessor.

6.1.6 1.0.6

- added `RenameFileProcessor`

6.1.7 1.0.5

- minor bug fixes.

6.1.8 1.0.4

- Switched to MIT License
- Added `stashed_value` to processors.

6.1.9 1.0.3

- Added support for `Wand` with `WandImageProcessor`.
- Made it compatible with Django 1.8
- Updated compiled JavaScript file.

6.1.10 1.0.2

- Introduced `pre_processor`.
- Made `UploadTo` serializible.
- Got rid of custom handlers.
- Minor bugfixes.

6.1.11 1.0.0

- Initial release

6.2 Authors

- Alexey Kuleshevich <lehins@yandex.ru> @lehins

6.3 License

```
The MIT License (MIT)

Copyright (c) 2015 Alexey Kuleshevich

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
```

(continues on next page)

(continued from previous page)

```
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

CHAPTER 7

Indices and tables

- genindex
- modindex

Index