django-recurrence Documentation

Release 1.10.1

django-recurrence developers

Sep 03, 2019

Contents

django-recurrence is a utility for working with recurring dates in Django.

It provides:

- Recurrence/Rule objects using a subset of rfc2445 (wraps `dateutil.rrule`) for specifying recurring date/times;
- `RecurrenceField` for storing recurring datetimes in the database;
- a JavaScript widget.

CHAPTER 1

Contents

1.1 Installation

1.1.1 Download the library

Firstly, you'll need to install `django-recurrence` from PyPI. The easiest way to do this is with pip:

```
pip install django-recurrence
```

Then, make sure `recurrence` is in your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
  ...
  'recurrence',
)
```

Supported Django and Python versions

Currently, django-recurrence supports Python 3.6 and Python 3.7.

django-recurrence works with Django 1.11, 2.1 and 2.2.

1.1.2 Set up internationalization

Note: This step is currently mandatory, but may be bypassed with an extra bit of javascript. See #47 for details.

Using a translation of django-recurrence other than `en` requires that django-recurrence's JavaScript can access the translation strings. This is handled with Django's built in `javascript_catalog` view, which you must install by adding the following to your project `urls.py` file:

```
import django
from django.conf.urls import url
from django.views.i18n import JavaScriptCatalog

# Your normal URLs here...

# If you already have a js_info_dict dictionary, just add
# 'recurrence' to the existing 'packages' tuple.
js_info_dict = {
    'packages': ('recurrence', ),
}

# jsi18n can be anything you like here
urlpatterns += [
    url(r'^jsi18n/$', JavaScriptCatalog.as_view(), js_info_dict),
]
```

1.1.3 Configure static files

django-recurrence includes some static files (all to do with rendering the JavaScript widget that makes handling recurring dates easier). To ensure these are served correctly, you'll probably want to ensure you also have `django.contrib.staticfiles` in your `INSTALLED_APPS` setting, and run:

```
python manage.py collectstatic
```

Note: After collecting static files, you can use `{{ form.media }}` to include recurrence's static files within your templates.

1.2 Usage

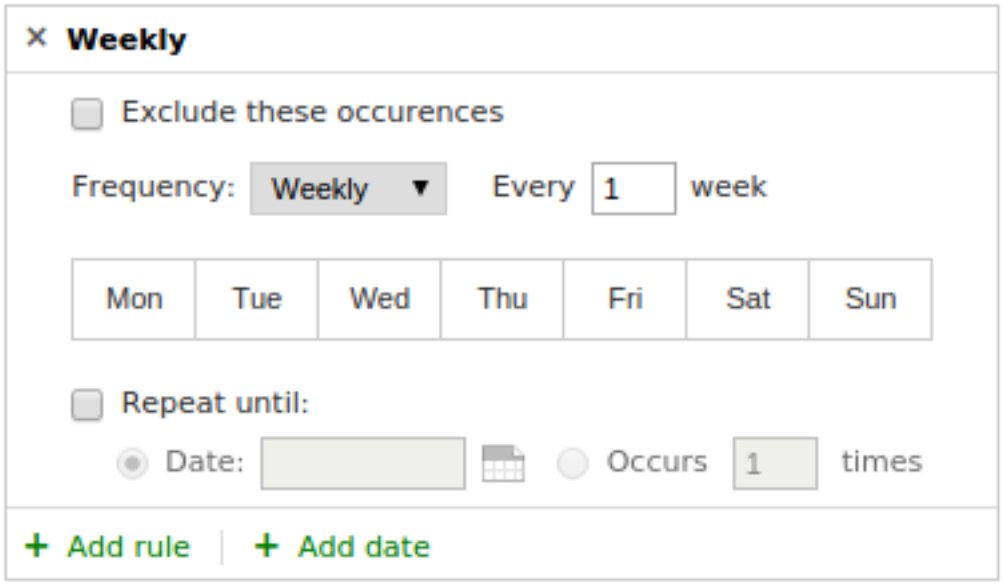
1.2.1 Getting started

Once you've *installed django-recurrence*, you'll generally want to start by using it in one of your models, which can be done like this:

```
from recurrence.fields import RecurrenceField

class Course(models.Model):
    title = models.CharField(max_length=200)
    recurrences = RecurrenceField()
```

If you use the `Course` model in Django's administrative interface, or in any forms, it should be rendered with a pretty form field, which makes selecting relatively complex recurrence patterns easy.



Using this form it's possible to specify relatively complex recurrence rules - such as an event that happens every third Thursday of the month, unless that Thursday happens to be the 21st of the month, and so on.

1.2.2 Form Usage

```
from django import forms
from .models import Course

class CourseForm(forms.ModelForm):
   class Meta:
      model = Course
      fields = ('title', 'recurrences',)
```

Note: Be sure to add `{{ form.media }}` to your template or statically link recurrence.css and recurrence.js.

```
<form method="POST" class="post-form">
    {% csrf_token %}
    {{ form.media }}
    {{ form }}
    <button type="submit">Submit</button>
</form>
```

1.2.3 Using `RecurrenceField`

Getting occurrences between two dates

Once you've created a model with a `RecurrenceField`, you'll probably want to use it to figure out what dates are involved in a particular recurrence pattern.

Note: Whether you want to use `between` or *occurrences* will depend on what sort of rules you're dealing with. In reality, a model like *Course* probably has rules like "every Thursday for 10 weeks", so `occurrences` will work fine, since the rules have natural limits. For other uses (e.g. find me every date this year for a club that runs every Wednesday), you'll want to use `between`.

`between` takes two dates (the start and end date), and returns a list of `datetime` objects matching the recurrence pattern between those dates. It is used like this (using the `Course` model from above):

```
from datetime import datetime
from myapp.models import Course


course = Course.objects.get(pk=1)
course.recurrences.between(
    datetime(2010, 1, 1, 0, 0, 0),
    datetime(2014, 12, 31, 0, 0, 0)
)
```

This won't include occurrences if they occur on the start and end dates specified. If you want to include those, pass in `inc` like this:

```
course.recurrences.between(
    datetime(2010, 1, 1, 0, 0, 0),
    datetime(2014, 12, 31, 0, 0, 0),
    inc=True
)
```

Warning: Slightly confusingly, `between` will only return you dates after the current date, if used as above (provided those dates fall between the two first parameters to `between`). Read on for how to get all the occurrences between two dates.

To get all the occurrences between two dates (including dates that are *before* the current time, but *after* the provided start date), you'll also need to set `dtstart`, like this:

```
course.recurrences.between(
    datetime(2010, 1, 1, 0, 0, 0),
    datetime(2014, 12, 31, 0, 0, 0),
    dtstart=datetime(2010, 1, 1, 0, 0, 0),
    inc=True
)
```

That will get you all occurrences between 1st January 2010, and 31st December 2014, including any occurrences on 1st January 2010 and 31st December 2014, if your recurrence pattern matches those dates.

The effective starting date for any recurrence pattern is essentially the later of the first argument and `dtstart`. To minimize confusion, you probably want to set them both to the same value.

Warning: Note that per default `dtstart` will be the first occurence in your list if specified, according to RFC 2445. This practice deviates from how `dateutil.rrule` handles `dtstart` and can therefore lead to confusion. Read on for how you can control this behavior for your own recurrence patterns.

To switch off the automatic inclusion of `dtstart` into the occurence list, set `include_dtstart=False` as an

argument for the `RecurrenceField` whose behavior you want to change:

```
class Course(models.Model):
    title = models.CharField(max_length=200)
    recurrences = RecurrenceField(include_dtstart=False)
```

With this change any `dtstart` value will only be an occurence if it matches the pattern specified in `recurrences`. This also works for instantiating `Recurrence` objects directly:

```
pattern = recurrence.Recurrence(
   rrules=[recurrence.Rule(recurrence.WEEKLY, byday=recurrence.MONDAY)],
   include_dtstart=False).between(
      datetime(2010, 1, 1, 0, 0, 0),
      datetime(2014, 12, 31, 0, 0, 0),
      dtstart=datetime(2010, 1, 1, 0, 0, 0),
      inc=True
   )
)
```

Getting all occurrences

`occurrences` is particularly useful where your recurrence pattern is limited by the rules generating occurrences (e.g. "every Tuesday for 10 weeks", or "every Tuesday until 23rd April 2014").

You can get a generator which you can iterate over to get all occurrences using `occurrences`:

```
dates = course.recurrences.occurrences()
```

You can optionally provide `dtstart` to specify the first occurrence, and `dtend` to specify the final occurrence.

You can index into the returned object, to (for example) get the first session of our course model:

```
dates = course.recurrences.occurrences()
first_instance = dates[0]
```

Warning: Looping over the entire generator returned by example above might be extremely slow and resource hungry if `dtstart` or `dtend` are not provided. Without `dtstart`, we implicitly are looking for occurrences after the current date. Without `dtend`, we'll look for all occurrences up to (and including) the year 9999, which is probably not what you want. The the code above counts all occurrences of our course from tomorrow until 31st December, 9999.

Counting occurrences

The function `count` works fairly similarly:

```
course.recurrences.count()
```

It is roughly equivalent to:

```
len(list(course.recurrences.occurrences()))
```

Note the warning in *occurrences* before using `count` (or converting the generator returned by `occurrences()` to a list), if you are not providing both `dtstart` and `dtend`.

Getting the next or previous occurrences

If you want to get the next or previous occurrence in a given pattern, you can use `after` or `before`, respectively. As with `between`, you can choose whether you want to be inclusive of the `datetime` passed in by setting `inc`. If no next or previous occurrence exists, `None` is returned.

```
course = Course.objects.get(pk=1)

# Get the first course on or after 1st January 2010 (this won't do
# quite what you expect)
course.recurrences.after(
    datetime(2010, 1, 1, 0, 0, 0),
    inc=True
)
```

As with `between`, if you don't specify a `dtstart`, it will implicitly be the current time, so the above code will, to be more precise, give you the first course on or after 1st January 2010, or on or after the current date, whichever is later. Since you probably don't want that behaviour, you'll probably want to specify `dtstart`, as follows:

```
course = Course.objects.get(pk=1)

# Get the first course on or after 1st January 2010
course.recurrences.after(
    datetime(2010, 1, 1, 0, 0, 0),
    inc=True,
    dtstart=datetime(2010, 1, 1, 0, 0, 0),
)
```

For similar reasons, using `before` really requires that `dtstart` is provided, to give a start date to the recurrence pattern. This makes some sense if you consider a recurrence pattern like "every Monday, occurring 5 times". Without `dtstart`, it's unclear what `before` should return - since it's impossible to know whether the pattern has started, and if so when. For example, if it started 5 years ago, `before` should return a date approximately 5 years ago, whereas if it started two weeks ago, `before` should return the last `Monday` (or the provided date, if `inc` is True, and the provided date is a Monday).

Getting textual descriptions of patterns

Recurrence patterns can have multiple rules for inclusion (e.g. every week, on a Tuesday) and exclusion (e.g. except when it's the first Tuesday of the month), together with specific dates to include or exclude (regardless of whether they're part of the inclusion or exclusion rules).

You'll often want to display a simple textual description of the rules involved.

To take our `Course` example again, you can get access to the relevant inclusion rules by accessing the `rrules` member of the `RecurrenceField` attribute of your model (called `recurrences` in our example, though you can call it whatever you like), and to the exclusion rules by accessing the `exrules` member. From there you can get textual descriptions, like this:

```
course = Course.objects.get(pk=1)
text_rules_inclusion = []

for rule in course.recurrences.rrules:
    text_rules_inclusion.append(rule.to_text())
```

Similar code would work equally well for `exrules`.

1.2.4 Generating Recurrences Programmatically

Rule and Recurrence

To create recurrence objects, the two main classes you'll need are `Rule` and `Recurrence`.

`Rule` specifies a single rule (e.g. "every third Friday of the month"), and a `Recurrence` is a collection of rules (some of which may be inclusion rules, others of which may be exclusion rules), together with date limits, and other configuration parameters.

For example:

```
from datetime import datetime
import recurrence


myrule = recurrence.Rule(
    recurrence.DAILY
)

pattern = recurrence.Recurrence(
    dtstart=datetime(2014, 1, 2, 0, 0, 0),
    dtend=datetime(2014, 1, 3, 0, 0, 0),
    rrules=[myrule, ]
)
```

You can then generate a set of recurrences for that recurrence pattern, like this:

```
>>> list(mypattern.occurrences())
[datetime.datetime(2014, 1, 2, 0, 0), datetime.datetime(2014, 1, 3, 0, 0)]
```

Exclusion Rules

You can specify exclusion rules too, which are exactly the same as inclusion rules, but they represent rules which match dates which should *not* be included in the list of occurrences. Inclusion rules are provided to the `Recurrence` object using the kwarg `rrules`, and exclusion rules are provided to the `Recurrence` object using the kwargs `exrules`.

Adding or Excluding Individual Dates

Similarly, you can specify individual dates to include or exclude using `rdates` and `exdates`, both of which should be a list of `datetime.datetime` objects.

```
from datetime import datetime
import recurrence

pattern = recurrence.Recurrence(
```

(continues on next page)

(continued from previous page)

```
    rdates=[
        datetime(2014, 1, 1, 0, 0, 0),
        datetime(2014, 1, 2, 0, 0, 0),
    ]
)
```

1.3 Contributing

Contributions to django-recurrence are very welcome - whether in the form of bug reports, feature requests, or patches. Bug reports and feature requests are tracked on our GitHub issues page.

If you want to make changes to django-recurrence, you'll need to fork our GitHub repository, make any changes you want, and send us a pull request. Feel free to file an issue if you want help getting set up.

1.3.1 Running the tests

The easiest way to run the tests is to run:

```
make testall
```

from the root of your local copy of the django-recurrence repository. This will require that you have tox installed. If you don't have tox installed, you can install it with `pip install tox`. Running all the tests also requires that you have Python 2.6, Python 2.7, Python 3.3 and Python 3.4 installed locally.

This will run tests against all supported Python and Django versions, check the documentation can be built, and will also run `flake8`, an automated code-linting tool.

If that sounds like too much work, feel free to just run tests on whatever your local version of Python is. You can do this by running:

```
pip install -r requirements_test.txt  ! You only need to run this once
make test
```

If you want to see what our code coverage is like, install everything in `requirements_test.txt` (as shown above), then run:

```
make coverage
```

1.3.2 Working with the documentation

Our documentation is written with Sphinx, and can be built using:

```
tox -e docs
```

Once this command is run, it'll print out the folder the generated HTML documentation is available in.

1.4 Changelog

1.4.1 1.10.1

- Update path to jQuery to match the one Django admin provides (#148).

1.4.2 1.10.0

- Fixes and official support for Django 2.1 and 2.2 (#143, #142);
- Remove support for Python 2.7 and 3.5, remove support for Django 2.0 (#145).

1.4.3 1.9.0

- Fix for using the recurrence widget in admin inlines (#137).

1.4.4 1.8.2

- Minor fix for Django 2.0 (#134);
- Minor packaging fix (#135).

1.4.5 1.8.1

- Bad release, do not use.

1.4.6 1.8.0

This release contains two backwards incompatible changes - please read the notes below for details.

- django-recurrence now returns timezone aware `datetime` objects in most cases (#130). If `USE_TZ` is `True` (it defaults to off with a stock Django install) then you'll now get timezone aware `datetime` objects back from django-recurrence. If you have `USE_TZ` as `True`, and you don't want this behaviour, you can set `RECURRENCE_USE_TZ` to `False`, but please let us know (via GitHub issues) that you wanted to opt out, so we can understand what your use case is.
- `RecurrenceField` instances without `required=False` will now require at least one rule or date. This change is intended to bring django-recurrence into line with how custom fields should behave. If you don't want to require at least one rule or date, just set `require=False` on your field (#133).
- Improvements to avoid inline styles (#85);
- Handle changes to `javascript_catalog` in Django 2 (#131).

1.4.7 1.7.0

- Drop official support for Django versions 1.7, 1.8, 1.9, 1.10;
- Fixes for saving `None` into a `RecurrenceField` causing a `TypeError` (#89, #122);
- Drop official support for Python 3.3 and Python 3.4;

- Provisional support for Python 3.7 (only for Django 2.0 and up);
- Ensure use of `render` on Django widgets always passes the `renderer` argument, to ensure support for Django 2.1 (#125);
- Django 2.0 compatibility fix for usage of django-recurrence with Django REST framework (#126).

1.4.8 1.6.0

- Fixes for Python 3 (#105);
- Support for Django 2.0 (#109, #110);
- Switch back a couple of instances of `DeserializationError` to `ValidationError` (#111);
- Switch around how we set dates in the date selector widget to avoid issues with short months (#113).

1.4.9 1.5.0

- Add Slovakian translations (#98);
- Add support for events occurring at a fixed point before the end of the month - e.g. the second last Tuesday before the end of the month (#88);
- Add minor style changes to make django-recurrence compatible with Wagtail (#100);
- Allow changing the behaviour of generating recurrences on `dtstart` by default. You can opt in to this by setting `include_dtstart=False` on your `RecurrenceField` (#93);
- Ensure broken values raise `DeserializationError` where expected (#103).

1.4.10 1.4.1

- Make PO-Revision-Date parseable by babel (#75);
- Update installation notes to cover Django 1.10 (#74);
- Add German translation (#77);
- Add Brazilian translation (#79);
- Ensure the migrations are included when installing (#78);
- Fix order of arguments to *to_dateutil_rruleset* (#81).

1.4.11 1.4.0

- Improve our testing setup to also cover Python 3.5;
- Fixes for Django 1.10 (#69).

1.4.12 1.3.1

- Add Basque translations (#67).

1.4.13 1.3.0

- Drop official support for Django 1.4, Django 1.5, Django 1.6 and Python 2.6 (no changes have been made to deliberately break older versions, but older versions will not be tested going forward);
- Add official support for Django 1.8 and Django 1.9 (#62);
- Fix for a bug in `Rule` creation where the weekday parameter is an instance of `Weekday` rather than an integer (#57).

1.4.14 1.2.0

- Added an option for events to occur on the fourth of a given weekday of the month (#29);
- Fixed an off-by-one bug in the `to_text` method for events happening on a regular month each year (#30);
- Fixed a bug in the JavaScript widget where the date for monthly events on a fixed date of the month had the description rendered incorrectly if the day selected was more than the number of days in the current calendar month (#31);
- Added a French translation (#32) - this may be backwards incompatible if have overriden the widget JavaScript such that there is no `language_code` member of your recurrence object;
- Added a Spanish translation (#49);
- Added database migrations - running `python manage.py migrate recurrence --fake` should be sufficient for this version - nothing has changed about the database schema between 1.1.0 and 1.2.0;
- Fix broken tests for Django 1.4.

1.4.15 1.1.0

- Added experimental Python 3 support.
- Added extensive test coverage (from 0% to 81%).
- Added documentation (including this changelog).
- Removed `RecurrenceModelField` and `RecurrenceModelDescriptor`, which don't appear to have worked as expected for some time.
- Fixed a bug introduced in 1.0.3 which prevented the django-recurrence JavaScript from working (#27).
- Don't raise `ValueError` if you save `None` into a `RecurrenceField` with `null=False` (#22), for consistency with other field types.
- Make sure an empty recurrence object is falsey (#25).
- Fix a copy-paste error in `to_recurrence_object` which prevented exclusion rules from being populated correctly.
- Fix a typo in `create_from_recurrence_object` which prevented it working with inclusion or exclusion rules.
- Various other very minor bugfixes.