
Django-minipub Documentation

Release 1.8.1

Richard Barran

Sep 03, 2022

Contents

| | | |
|----------|--|-----------|
| 1 | Installation & configuration | 3 |
| 1.1 | Installation | 3 |
| 1.2 | Dependencies | 3 |
| 1.3 | Configure Your Django Settings | 3 |
| 2 | The Minipub model | 5 |
| 2.1 | Timestamps | 5 |
| 2.2 | The concept of ‘Live’ objects | 5 |
| 2.3 | ‘staff_preview’ property | 6 |
| 3 | Minipub views | 7 |
| 4 | Admin | 9 |
| 5 | Sitemap | 11 |
| 5.1 | If you have custom statuses. | 11 |
| 6 | Refining the concept of a ‘Live’ object | 13 |
| 6.1 | Models | 13 |
| 6.2 | Views | 13 |
| 7 | Contributing to Django-minipub | 15 |
| 7.1 | Example project | 15 |
| 7.2 | Coding style | 15 |
| 7.3 | Unit tests | 15 |
| 7.4 | Documentation | 15 |
| 8 | Indices and tables | 17 |
| | Python Module Index | 19 |
| | Index | 21 |

Contents:

Installation & configuration

1.1 Installation

The easiest way to install Minipub is to get the latest version from [PyPi](#):

```
pip install django-minipub
```

You can also live dangerously and install the latest code directly from the Github repository:

```
pip install -e git+https://github.com/richardbarran/django-minipub.git#egg=django-  
↪minipub
```

This code should work ok - like [Django](#) itself, the master branch should be bug-free. Note however that you will get far better support if you stick with one of the official releases!

1.2 Dependencies

The following dependencies will be installed automatically if required:

- Django.
- Django-model-utils.

1.3 Configure Your Django Settings

Add minipub to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (  
    # ...other installed applications,  
    'minipub',  
)
```

You are now ready to use minipub in your code.

CHAPTER 2

The Minipub model

```
from minipub.models import MinipubModel

class Article(MinipubModel):
    ...
```

MinipubModel is an abstract model that provides the following 3 fields:

- status: a list of choices; default is ‘draft’ or ‘published’.
- start: a start date; defaults to date of publication.
- end: an end date; optional.

2.1 Timestamps

MinipubModel also adds the following fields that get auto-updated and should not be manually modified: `created`, `modified` and `status_changed`.

2.2 The concept of ‘Live’ objects

Objects are usually considered ‘live’ **if** they are ‘published’ **and** between the start and end dates - this is usually sufficient for them being available to display in the public website.

`live()` methods are available both as a chainable filter on a queryset, and as an instance method. For example, if you have an `Article` model that uses `MinipubModel`:

```
my_articles = Article.objects.live()
```

or

```
can_be_viewed = article1.live()
```

2.2.1 Extra statuses

Models can have more statuses than draft, published - [see here for more details](#).

2.2.2 Sitemaps

If you have defined a sitemap.xml, refer also to the [sitemaps page](#).

2.3 ‘staff_preview’ property

`MinipubModel.staff_preview()`

Helper property - says if this object is being previewed by a member of staff.

Can be used when displaying objects in the website - members of staff will see all objects, using this property in the template can help for attaching a message/custom CSS saying that this object is being previewed.

For example, in the `example_project` we have this snippet:

```
{% if article.staff_preview %}
    <div class="label label-warning">This is a preview</div>
{% endif %}
```

CHAPTER 3

Minipub views

```
from minipub.views import MinipubArchiveIndexView, MinipubYearArchiveView,   
↳MinipubDetailView  
  
class ArticleArchiveView(MinipubArchiveIndexView):  
    ...
```

Minipub provides a few basic views to get you started - but you are encouraged to look at the source code. The important thing to note is the `GetQuerysetMixin` mixin - you can use this with most of Django's List- and Detail-class based-views to easily integrate minipub.

For example, in `minipub' views.py`, here is the source code for defining a detail view:

```
class MinipubDetailView(GetQuerysetMixin, DetailView):  
    pass
```


We have a `MinipubAdmin` class that basically gives you the code for displaying 2 fieldsets:

1. The fieldset that controls publication:

Publication

You can control here the publication of this object.
 An object is only 'viewable' if its status is 'published', and if today's date is after the start date. If the end date is entered, the object will no longer be viewable after that date.
Note: when you are logged in to the 'admin', you can preview this object in the website, even if it's not published.

Status:
Start date: Today |
End date: Today |

2. And the 'status' fieldset; these fields are read-only, and exist just for information:

Timestamps (Hide)

When this record was created, last modified, and when the status was last changed.

Created: 10 Sep 2014, 12:57 a.m. **Modified:** 10 Sep 2014, 1:02 a.m.
Status changed: 10 Sep 2014, 12:57 a.m.

And here is an example of using them:

```
from minipub.admin import MinipubAdmin

class ArticleAdmin(MinipubAdmin):

    # Other code...

    fieldsets = (
        (None, {
            'fields': ('some', 'other', 'fields',)
        }),
        MinipubAdmin.PUBLICATION_FIELDSET,
        MinipubAdmin.TIMESTAMP_FIELDSET
    )
```


We have a `MinipubSitemap` class that provides a small amount of boilerplate to make creating a sitemap easier. Here is an example of how to use it:

```
from minipub.sitemaps import MinipubSitemap

from .models import Article

class NewsSitemap(MinipubSitemap):
    model = Article
```

Basically, all you'll need to do is set the model to use in the sitemap - in this example, it's `Article`.

5.1 If you have custom statuses...

If you have more statuses than the default 'draft', 'published', and you have different sets of views to display these - you will also need extra sitemap classes.

You can add a `minipub_live` attribute to your sitemap class - exactly the same way as you will have done for your extra views. For example:

```
from minipub.sitemaps import MinipubSitemap

from .models import Article

class NewsArchivesSitemap(MinipubSitemap):
    """Sitemap for the archived articles."""
    model = Article
    minipub_live = ('archived',)
```

Refining the concept of a ‘Live’ object

Let’s imagine that your articles can be published, but that at some point in time you will want to manually move them to an ‘archived’ section in the website.

6.1 Models

First, you will define in your model the complete list of statuses available:

```
class MyCustomModel(MinipubModel):  
  
    STATUS = Choices('draft', 'published', 'archived')
```

And you’ll also tweak your `get_absolute_url()` to handle the new options:

```
def get_absolute_url(self):  
    if self.status == self.STATUS.archived:  
        return reverse('the_url_for_the_archived_page', kwargs={'slug': self.slug})  
    else:  
        return reverse('the_url_for_the_published_page', kwargs={'slug': self.slug})
```

6.2 Views

Secondly, in your `views.py` you will have 2 sets of views:

- the views for the main pages.
- the views for the archived pages.

And the urls would look something like:

```
/news/  
/news/<article slug>/  
/archives/  
/archives/<article slug>/
```

In the views, you will split out the articles to be displayed in the main section:

```
class ArticleDetailView(MinipubDetailView):  
    model = Article  
    context_object_name = 'article'  
    minipub_live = ('published',)
```

And then the articles to show in the archives section:

```
class ArticleArchivesDetailView(MinipubDetailView):  
    model = Article  
    context_object_name = 'article'  
    minipub_live = ('archived',)
```

We have added a new attribute - `minipub_live`. This will override the `STATUS` defined on the model.

Contributing to Django-minipub

Contributions are always very welcome. Even if you have never contributed to an open-source project before - please do not hesitate to offer help. Fixes for typos in the documentation, extra unit tests, etc. . . are all welcome.

7.1 Example project

Django-minipub includes an example project under `/example_project/` to get you quickly ready for contributing to the project - do not hesitate to use it!

You'll probably also want to manually install [Sphinx](#) if you're going to update the documentation.

7.2 Coding style

No surprises here - just try to [follow the conventions used by Django itself](#).

7.3 Unit tests

Including unit tests with your contributions will earn you bonus points, maybe even a beer. So write plenty of tests, and run them from the `/example_project/` with a `python manage.py test`.

There's also a [Tox](#) configuration file - so if you have tox installed, run `tox` from the `/example_project/` folder, and it will run the entire test suite against all versions of Python and Django that are supported.

7.4 Documentation

Keeping the documentation up-to-date is very important - so if your code changes how Django-minipub works, or adds a new feature, please check that the documentation is still accurate, and update it if required.

We use [Sphinx](#) to prepare the documentation; please refer to the excellent docs on that site for help.

Note: The CHANGELOG is part of the documentation, so don't forget to update it!

CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`

m

- `minipub.admin`, 9
- `minipub.models`, 5
- `minipub.sitemaps`, 11
- `minipub.views`, 7

M

`minipub.admin` (*module*), 9
`minipub.models` (*module*), 5
`minipub.sitemaps` (*module*), 11
`minipub.views` (*module*), 7

S

`staff_preview()` (*minipub.models.MinipubModel*
method), 6