
django-generic-flatblocks Documentation

Release 1.0

Martin Mahner

Apr 05, 2017

Contents

1	Installation	3
2	Quickstart	5
3	Detailed usage	7
3.1	The arguments in detail:	7
4	Create your own content node	9
5	Contributed content nodes	11

If you want to add tiny snippets of text to your site, manageable by the admin backend, you would use either [django-chunks](#) or [django-flatblocks](#). However, both of them have one problem: you are limited to a predefined content field; a “text” field in chunks and a “title” and “text” field in flatblocks.

django-generic-flatblocks solves this problem as it knows nothing about the content itself. You *attach* your hand made content node (a simple model) where you can define any fields you want.

Contents:

CHAPTER 1

Installation

This package is available through the python package index, pypi. You can install the latest version by:

```
pip install django-generic-flatblocks
```

Add the module to your `INSTALLED_APPS` in your settings:

```
INSTALLED_APPS = (  
    ...  
    'django_generic_flatblocks',  
    'django_generic_flatblocks.contrib.gblocks', # Optional sample models  
)
```

Make sure that `django.core.context_processors.request` was added to your `TEMPLATE` options:

```
TEMPLATES = [  
    {'BACKEND': 'django.template.backends.django.DjangoTemplates',  
      'OPTIONS': {  
          'context_processors': [  
              'django.template.context_processors.request',  
              ...  
          ]  
      }  
    }  
]
```

(*Optional*) Define the url prefix to your `contrib.admin` installation in the setting `ADMIN_URL_PREFIX`. Most commonly this is `/admin/`. Beware the trailing slash.

Migrate the database schemas:

```
./manage.py migrate
```

See [Quickstart](#) for a quick demonstration or [Detailed usage](#) for a detailed integration.

CHAPTER 2

Quickstart

You can join unlimited of slug-strings or context-variables to one slug. Most commonly you will do this if you need to use the users LANGUAGE_CODE in your slug, to have different content nodes for every language:

```
{% load generic_flatblocks %}
{% gblock "website","title",LANGUAGE_CODE for "gblocks.Title" %}
```

The slug can also be a context variable:

```
{% with "website_teaser" as my_slug %}
    {% gblock my_slug for "gblocks.Text" %}
{% endwith %}
```

You can render each generic block with a template of your choice:

```
{% gblock "website_urgent_notice" for "gblocks.Text" with "urgent.html" %}
```

You can pass an integer as slug. In this case, generic-flatblocks will fetch the model instance with the primary key you named in slug. Basically this is a `{% include %}` tag on model level:

```
{% gblock 1 for "auth.user" with "current_user.html" %}
```

You can store the related object directly in the context using the “into” argument. This is the quickest way to display any model. The “for” and “as” arguments are ignored:

```
{% gblock 1 for "auth.user" into "the_user_object" %}
<p>The first user is {{ the_user_object.username }}!</p>
{% if the_user_object_admin_url %}<a href="{{ the_user_object_admin_url }}">edit</a>{
→% endif %}
```

Let’s create an flatblock with a “as” argument. We publish this block at the end of this page in a variable called FOOTER:

```
{% gblock "footer" for "gblocks.Text" as "FOOTER" %}

{{ FOOTER }}
```



Detailed usage

First of all, in every template you want to use generic-flatblocks, load the templatetags library:

```
{% load generic_flatblocks %}
```

Then define a content node using the `gblock` templatetag:

```
{% gblock "unique_slug" for "applabel.modelname" with "render/with/template.html" as  
↪"variable" %}
```

The arguments in detail:

“unique_slug” (required):

The slug argument defines under which *key* the content is stored in your database. You can define as many slugs as you want, just use a comma as separator. You can use context-variables as well. Examples:

```
"homepage headline" becomes "homepage_headline"  
"homepage", "headline" becomes "homepage_headline"  
"homepage_title", LANGUAGE_CODE becomes "homepage_title_en" (depends on the users_  
↪ locale code)  
"user", user.pk becomes "user_1" (depends on the primary key of the currently logged_  
↪ in user)
```

You can pass an *integer* as the slug. This will cause the templatetag to fetch the model named in *for* with the primary key you named in *slug*. Example:

```
{% gblock 1 for "auth.user" with "path/to/template.html" %}
```

This will fetch the `auth.User` with the primary key 1 and renders this model object with the template “path/to/template.html”. In this case, the `generic_object` is `None`. Basically this is a `{% include %}` tag on model level. This can also be a context variable.

for “applabel.modelname” (required):

The *for* argument defines, what content-node (model) will be used to store and display the content. The format is *appname.modelname*. For some contributed content-nodes see *Contributed content nodes* below. This argument can be a context-variable.

with “template_path” (optional):

You can define a template that is used for rendering the content node. If you do not provide any template, the default template `<applabel>/<modelname>/flatblock.html` is used. This argument can be a context-variable.

In this template are all context-variables from the *parent* template available plus some extra variables:

- `object`: This variable is the model-instance for the generic block.
- `generic_object`: This variable is the model-instance for the generic content object itself. Mostly you don't need this.
- `admin_url`: A URL to the change view of the current object. This variable is `None` if the current user has no change permissions for the object.

as “variable name” (optional):

If you provide a variable name, the *rendered content node* is stored in it. Otherwise it's displayed directly. This argument can be a context-variable.

into “variable_name” (optional):

If you provide a variable name, the *related object* is stored in it. No template rendering is done. The *with* and the *as* arguments are ignored. This argument can be a context-variable.

After calling the `gblock` templatetag, you have the same variables available as in the *with* template:

- `variable_name`: This variable is the model-instance for the generic block.
- `variable_name + "_generic_object"`: This variable is the model-instance for the generic content object itself. Mostly you don't need this.
- `variable_name + "_admin_url"`: A URL to the change view of the current object. This variable is `None` if the current user has no change permissions for the object.

This is the quickest way to display any model instance or content-node directly without creating a template:

```
{% gblock 1 for "auth.User" into "theuser" %}
The first user is {{ theuser.username }}! (<a href="{{ theuser_admin_url }}">edit</a>)
```

would be rendered as:

```
The first user is johndoe! (<a href="/admin/auth/user/1/">edit</a>)
```

Note: If you have `settings.TEMPLATE_DEBUG` set to `True` and your related object does not exist, the templatetag will raise a `ObjectNotFound` exception. It will fail silently if you set `settings.TEMPLATE_DEBUG` to `False` and return an (empty, not saved) instance of the related model.

Create your own content node

A content node is a simple django-model. No quirks. If you want to use a title and a textfield as your content-node, define a new model `Entry` in your application `myproject`:

```
from django.db import models
from django.contrib import admin

class Entry(models.Model):
    title = models.CharField(max_length=255, blank=True)
    content = models.TextField(blank=True)

    def __unicode__(self):
        return self.title

admin.site.register(Entry)
```

Important: `django-generic-flatblocks` creates an empty content-node upon first request, so make sure each field has either its default value or allow `blank=True`. Don't forget to register your Model in the admin backend, if you want to edit it there.

Then create a template `myproject/entry/flatblock.html` in your template directory. This template is the default template to render the content node, if you do not provide a unique template for it (*with* argument).

In this template are all context-variables from the *parent* template available plus some extra variables. See arguments/with above for details.

A common template source for the content node would be:

```
<h1>{{ object.title }}</h1>
{{ object.content|safe }}

{% if admin_url %}<a href="{{ admin_url }}">edit this</a>{% endif %}
```

In your templates, create a new content node using the `templatetag`:

```
{% gblock "about_me" for "myproject.Entry" %}
```

For some pre defined nodes see *Contributed content nodes*

Contributed content nodes

django-generic-flatblocks comes with some very commonly used content-nodes. They are not installed by default. To do so, insert `django_generic_flatblocks.contrib.gblocks` to your `INSTALLED_APPS` in your settings and resync your database: `./manage.py syncdb`.

The contributed content nodes are:

- **gblocks.Title**: A CharField rendered as a `<h2>` Tag.
- **gblocks.Text**: A TextField rendered as html paragraphs. (This is what django-chunks provides)
- **gblocks.Image**: A ImageField rendered as `` Tag.
- **gblocks.TitleAndText**: A CharField and a TextField. (This is what django-flatblocks provides)
- **gblocks.TitleTextAndImage**: A CharField, TextField and ImageField

So if you want to display a title and textfield, use this templatetag for example:

```
{% gblock "about_me" for "gblocks.TitleAndText" %}
```