Stoat Documentation

Release 0.4.0

Steve Losh and Dumbwaiter Design

May 30, 2012

CONTENTS

Stoat is a sleek, lightweight, pluggable CMS app for Django, built to work with Grappelli.

Stoat is like flatpages on steroids. No, scratch that, Stoat isn't that bulky. It's more like flatpages on a good exercise routine.

If you want to know why you should use Stoat instead of one of the many other options, check out the *Why Stoat?* page.

At this moment Stoat is very much alpha software – use it at your own risk!

The source is on BitBucket and on GitHub. Pull requests are accepted from either site.

Report issues on the GitHub issue tracker.

CHAPTER

ONE

DOCUMENTATION

1.1 Why Stoat?

Stoat was born at Dumbwaiter Design out of a desire for a simple CMS app that we could use for client sites.

We wanted something with enough features that didn't take over the site and didn't break every time a new version of Django/Grappelli/Filebrowser was released.

There are many other CMS apps for Django out there, and they are plump with many juicy features.

We wanted something sleeker.

1.1.1 What Stoat Has

Here are the currently implemented features:

- Arbitrary URLs.
- Few dependencies. At the moment treebeard and django-templatetag-sugar are the only ones and both install cleanly with pip.
- Support for (read: doesn't break) the `APPEND_SLASH` setting.
- Multiple templates for pages, with custom fields for each template.
- Multiple field types, like Filebrowser Image fields.
- Drag and drop reordering of pages, including making pages children of other pages.
- South support for migrating the Stoat database between versions.
- Built for and compatible with the latest version of Django (1.3), Grappelli and Filebrowser.
- Support for django-ckeditor for rich text editing.
- A few simple helpers for creating navigation menus.
- Support for inline models attached to Pages.
- Built-in support for Haystack.
- A small test suite.

1.1.2 What Stoat Doesn't Have Yet

Here's what we're planning on adding in the future:

- A more extensive test suite.
- More field types.
- More documentation.
- Publishing control. But it will be configurable and opt-in!

1.1.3 What Stoat Will Never Have

Here are the things we don't want to have in Stoat:

- i18n support. It adds complexity and many sites simply don't need it.
- Tagging. Pages don't need to be tagged, and half of the Django tagging apps are broken anyway.
- Redirects. The built-in redirects app handles these perfectly fine.
- "In-site" editing. The admin interface is for editing, the site is for using.
- Complicated permissions. Stoat embraces Django's admin permission system and doesn't try to add complexity on top of it.
- Revision control of pages (though some other apps that add this may happen to work with Stoat).

If you need some or all of these you then Stoat simply isn't for you. You should look at one of the many other Django CMS apps around.

1.2 Installation

Installing Stoat is just like installing any other Django app.

First, install the library (preferably into a virtualenv):

```
pip install -e hg+https://bitbucket.org/sjl/stoat@v0.1.1#egg=stoat
```

Add `stoat` to your `INSTALLED_APPS` setting:

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

Finally, add the following line at the end of your `MIDDLEWARE_CLASSES` setting:

```
MIDDLEWARE_CLASSES = (
    ...
    'stoat.middleware.StoatMiddleware',
)
```

If you've customized your admin dashboard with Grappelli's dashboard tools, you'll need to add Stoat to the dashboard to be able to add pages and such:

```
self.children.append(modules.ModelList(
    'Stoat',
    ('stoat.*',),
    column=1,
    css_classes=['collapse', 'open'],
))
```

Now that you've got Stoat installed you'll want to take a look at *Usage* to learn how to use Stoat.

1.3 Basic Usage

The first step to using Stoat is defining the templates you want to use for pages.

1.3.1 Defining Templates

You define the templates Stoat will use in your Django settings. Here's a quick example to get you started:

```
STOAT_TEMPLATES = {
    'Default': ['default.html', [
        ['Heading',       'char'],
        ['Body',          'ckeditor', { 'required': True }],
        ['Sidebar Body',  'text'],
        ['Sidebar Links', 'inline', { 'import': 'sidebar.admin.SidebarLinkInline' }],
    ]],
    'Product': ['pages/product.html', [
        ['Price',       'int', { 'required': True }],
        ['Description', 'text'],
        ['Image',       'img'],
        ['Salesperson', 'fk', { 'app': 'auth', 'model': 'User' }],
    ]],
}
STOAT_DEFAULT_TEMPLATE = 'Default'
```

The `STOAT_TEMPLATES` setting contains a dictionary mapping template names to template definitions.

Each template definition consists of:

- A path
- A list of fields
- A dictionary of configuration options (optional)

The path is a normal Django template path, like you might use with `render_to_response`.

Each field is a list containing a name and a field type. The field name is how the field will be labels in the admin and referred to in your templates, and the field type is one of the supported types detailed in the next section.

The option dictionary maps options to values. Some options (such as `required`) can be used with any kind of field, while others are specific to a certain field type. See the Field Options section for more information on universal options, and the Field Types section for options specific to a single type.

There's also a `STOAT_DEFAULT_TEMPLATE` setting that you need to set to the name of the template that should be considered the default.

1.3.2 Field Types

The following types of fields are available for use.

Note: Stoat actually stores all field data as Text data. The field types only change the form field type used (and validated) in the admin.

bool

A basic Django BooleanField.

char

A basic Django CharField.

ckeditor

A CKEditor field. This requires that django-ckeditor be installed.

Options

- `config`: A string defining which of the `CKEDITOR_CONFIGS` should be used for this field.

decimal

A basic Django DecimalField.

email

A basic Django EmailField.

float

A basic Django FloatField.

fk

A foreign key to a model. When using this field you *must* provide both of its options.

Options

- `app`: A string containing the name of the app the model is from.
- `model`: A string containing the name of the model.

img

A FileBrowseField from django-filebrowser, with the `'Image'` type.

inline

An `inline` field allows you to create models that have a ForeignKey pointing at Stoat pages.

Check out the **:doc:'Inlines </inlines>'_** documentation for more information.

int

A basic Django IntegerField.

text

A basic Django CharField, but rendered as with a `<textarea>` widget.

url

A basic Django URLField (with `verify_exists` set to `False`).

1.3.3 Field Options

The following options can be used for any field.

`required`

Ensures that this field is not left blank when editing the page in the admin interface.

`help_text`

Defines help text for individual fields.

1.3.4 Further Usage

Next you'll want to check out the *Admin* documentation to learn how to use the admin interface for adding pages (it's simple), and the *Templating* documentation to learn how to create Django templates for use with Stoat.

1.4 Inlines

Stoat allows you to create your own models that have a ForeignKey field pointing to `stoat.models.Page` and display them as inlines on the `Page` admin.

This can be useful if you want to allow an arbitrary number of a certain kind of data to be attached to a `Page`. We'll use the idea of a "sidebar link" as an example.

1.4.1 Creating Models

The first step is to create the `SidebarLink` model and make sure it has a ForeignKey to `stoat.models.Page`.

IMPORTANT: You *must* use a string when specifying the model for the ForeignKey, and not try to import `stoat.models.Page` directly, otherwise you'll get circular imports.

Here's a sample of a simple `models.py` file for our `SidebarLink` example:

```
from django.db import models

class SidebarLink(models.Model):
    title = models.CharField(max_length=140)
    link = models.URLField(verify_exists=True)
    page = models.ForeignKey('stoat.Page')
```

1.4.2 Creating Admins

The next step is to create the inline admin class as normal.

Here's a sample `admin.py` file for our example:

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

class SidebarLinkInline(admin.TabularInline):
    model = SidebarLink
    extra = 1
```

Both `TabularInline` and `StackedInline` will work, and you can configure the inline however you like.

1.4.3 Configuring Stoat

To tell Stoat to use this inline, you add a field to your template definition in `settings.py`:

```
STOAT_TEMPLATES = {
    'Default': ['default.html', [
        ['Heading',       'char'],
        ['Body',          'ckeditor', { 'required': True }],
        ['Sidebar Links', 'inline', { 'import': 'sidebar.admin.SidebarLinkInline' }],
    ]],
}
```

The definition for our new inline on its own looks like this:

```
['Sidebar Links', 'inline', { 'import': 'sidebar.admin.SidebarLinkInline' }],
```

The field options *must* contain the full path to import the inline.

1.5 Using the Admin Interface

Stoat uses the standard Django admin interface (through Grappelli) with a few customizations to streamline page editing.

The process of adding and editing pages is slightly more complicated than adding and editing standard Django models. We think that in practice this extra complexity is worth it for the power and flexibility it makes possible.

1.5.1 Creating and Editing Pages

Creating a new Stoat page has three steps:

- Add the new page (selecting a template in the process) and save it.
- Edit the page to fill in the template fields that weren't available in the "add" step.
- Move the page to its desired location in the hierarchy.

1.5.2 Changing Page Templates

When you select a different template for a page something important happens:

Any data for fields that are not in the new template will be discarded.

This behavior allows us to keep the Stoat codebase simple and free of extra complexity.

This may seem awful at first, but in practice it's not a very painful sacrifice. We've found that for most sites you almost never need to switch templates after a page has been created.

1.5.3 Moving Pages

You've probably already noticed the `Move` buttons on the `Page` admin changelist. To shuffle pages around in the hierarchy you can simply drag and drop these buttons to their desired place.

One thing to note: pages cannot be dropped onto pages under themselves. This would be complex and nonobvious behavior if we allowed it, so Stoat simply chooses to make it impossible.

1.6 Configuration

Stoat aims for convention over configuration, but there are still a few settings required, and a few more available if you need them.

1.6.1 Required Settings

These two settings need to be set for Stoat to work correctly.

STOAT_TEMPLATES

This setting defines the templates available for use, as described in the *Templating Documentation*.

STOAT_DEFAULT_TEMPLATE

This setting defines the default template, as described in the *Templating Documentation*.

1.6.2 Optional Settings

These settings are completely optional.

STOAT_DEBUG

Set this to `True` if you're working on Stoat itself. It does the following:

- Show the `PageContent` model in the admin interface.
- Show the `Template` column in the Page admin listing page.

STOAT_HIDE_NAVIGATION

Set this to `True` to hide the `show_in_nav` field in the page admin.

1.7 Templating

The templates you make to use with Stoat are simply Django templates that take advantage of an extra variable.

1.7.1 The `Page` Variable

When Stoat renders a template it adds a `page` variable to the context. This variable has a few properties you'll want to use.

page.title

The title of the page as defined in the admin interface.

page.get_absolute_url

A normal Django `get_absolute_url` method that will return the page's URL.

page.parent

The page's parent.

page.fields

This property contains all of the fields you've defined in `STOAT_TEMPLATES`, with their names lowercased and every non-letter/number replaced by an underscore.

For example: look at the following `STOAT_TEMPLATES` setting:

```
STOAT_TEMPLATES = {
    'Default': ['default.html', [
        ['Heading',         'char'],
        ['Body',            'text'],
        ['Sidebar Heading', 'text'],
    ]],
    'Product': ['pages/product.html', [
        ['Price',       'int'],
        ['Description', 'text'],
        ['Image',       'img'],
        ['Image 2',     'img'],
    ]],
}
```

Here's what `pages/product.html` might look like:

```
{% extends "base.html" %}

{% block content %}
    <h1>{{ page.title }}</h1>

    <img class="product-image" src="{{ page.fields.image }}" />
    <img class="product-image" src="{{ page.fields.image_2 }}" />

    <p class="price">Price: ${{ page.fields.price }}</p>

    {{ page.fields.description|linebreaks }}
{% endblock %}
```

You can use `page.f` as a shortcut for `page.fields` if you'd like to save on some typing.

1.7.2 Navigation

Stoat contains two sets of helpers for building navigation menus in your templates: template tags and `Page` methods.

Every page has a `show_in_nav` option that determines whether they will be part of the lists returned by these helpers.

Page Methods

page.breadcrumbs

A list of the page's ancestors and itself. For example, imagine you have the following page layout:

```
About Us
|
+-> The Team
    |
    +-> Jimmy
```

```
    |
    +-> Timmy
```

For the "Timmy" page `page.breadcrumbs` will be `[<About Us>, <The Team>, <Timmy>]`.

Each item in the list is a normal page object.

Here's an example of creating a simple list of breadcrumbs in an HTML template:

```
<ul>
    {% for p in page.breadcrumbs %}
        <li {% if forloop.last %}class="active"{% endif %}>
            <a href="{{ p.get_absolute_url }}">{{ p.title }}</a>
            {% if not forloop.last %}
                &gt;
            {% endif %}
        </li>
    {% endfor %}
</ul>
```

page.nav_siblings

A list of the page's siblings, including itself.

page.nav_children

A list of the page's children.

page.nav_siblings_and_children

A nested list of the page's siblings (including itself) and their children. For example, imagine the following layout:

```
Products
|
+-> Guitars
|
+-> Drums

About Us
|
+-> Hours
|
+-> Return Policy
```

For the "Products" or "About Us" page `page.nav_siblings_and_children` will be:

```
[
    [<Products>, [
        <Guitars>,
        <Drums>,
    ]],
    [<About Us>, [
        <Hours>,
        <Return Policy>,
    ]],
]
```

This property can be useful if you're trying to build a two-level navigation list (possibly with Javascript dropdowns). Here's an example of building such a list:

```
<ul>
    {% for top_page, child_pages in page.nav_siblings_and_children %}
        <li>
            <a href="{{ top_page.get_absolute_url }}">{{ top_page.title }}</a>

            {% if child_pages %}
                <ul>
                    {% for child_page in child_pages %}
                        <li>
                            <a href="{{ child_page.get_absolute_url }}">{{ child_page.title }}</a>
                        </li>
                    {% endfor %}
                </ul>
            {% endif %}
        </li>
    {% endfor %}
</ul>
```

Template Tags

To use these template tags you'll need to `{% load stoat %}` first.

These tags do *not* require that there be a Stoat page at the current URL, so you can safely use them anywhere you like.

`nav_roots`

A list of all the root pages.

`nav_roots_and_children`

A nested list of all of the root pages and their children (similar in structure to page.nav_siblings_and_children).

1.8 Contributing

Contributing to Stoat is simple: send a pull request on BitBucket or on GitHub.

If you're fixing a bug: great!

If you're adding a feature you might want to read the *Why Stoat?* page to help understand what Stoat is and what features we want to include.