django-scribbler Documentation

Release 0.6.0

Mark Lavin

January 18, 2016

Contents

django-scribbler is an application for managing snippets of text for a Django website. Similar projects include django-flatblocks, django-chunks and django-pagelets. This project attempts to take some of the best concepts from those previous projects as well as focus on giving the users instant feedback inspired by Bret Victor's Inventing on Principle talk.

CHAPTER 1

Features

- Simple template tag for defining snippet blocks with default text
- Template tag for displaying and editing fields from arbitrary models
- Front-end editing of snippets with the powerful CodeMirror editor
- Live in-place preview of content while editing
- The full power of the Django template language in the snippet blocks
- Python 3 support

CHAPTER 2

Installation

django-scribbler requires Django >= 1.3 and Python >= 2.6. Starting with version v0.2 there will be experimental Python 3 support (3.2+). Using Python 3 requires using Django >= 1.5.

To install from PyPi:

```
pip install django-scribbler
```

CHAPTER 3

Documentation

Documentation on using django-scribbler is available on Read The Docs.

CHAPTER 4

License

django-scribbler is released under the BSD License. See the LICENSE file for more details.

CHAPTER 5

Contributing

If you think you've found a bug or are interested in contributing to this project check out django-scribbler on Github. A full contributing guide can be found in the online documentation.

If you are interested in translating django-scribbler into your native language you can join the Transifex project.

Development sponsored by Caktus Consulting Group, LLC.

CHAPTER 6

Contents

6.1 Getting Started

Below are the basic steps need to get django-scribbler integrated into your Django project.

6.1.1 Configure Settings

You need to include `scribbler` to your installed apps. django-scribbler requires `django.contrib.auth` which in turn requires `django.contrib.sessions` which are enabled in Django by default. You will also need to include a context processor to include the current request in the template context.

```
INSTALLED_APPS = (
    # Required contrib apps
    'django.contrib.auth',
    'django.contrib.sessions',
    # Other installed apps would go here
    'scribbler',
)

TEMPLATE_CONTEXT_PROCESSORS = (
    # Other context processors would go here
    'django.core.context_processors.request',
)
```

Note that `TEMPLATE_CONTEXT_PROCESSORS` is not included in the default settings created by `startproject`. You should take care to ensure that the default context processors are included in this list. For a list of default `TEMPLATE_CONTEXT_PROCESSORS` please see the official Django docs.

If you are using Django 1.8 then instead of `TEMPLATE_CONTEXT_PROCESSORS` you should configure `TEMPLATES['OPTIONS']['context_processors']`:

```
TEMPLATES = [ # example config untill 'context_processors' your config maydiffer
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                    # add required context processors here:
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
```

```
                # Other context processors would go here
            ],
            'debug': False,
        },
    },
],
```

Note that unlike `TEMPLATE_CONTEXT_PROCESSORS` from Django 1.7 `TEMPLATES` is already included in the default settings created by `startproject`. Django 1.8 also supports custom template engines but this is not supported at the moment by django-scribbler.

For the context processor to have any effect you need to make sure that the template is rendered using a RequestContext. This is done for you with the render shortcut.

django-scribbler aggressively caches the scribble content. By default the scribble content is cached for 12 hours. You have the option to configure this cache timeout with the `SCRIBBLER_CACHE_TIMEOUT` setting. The value should be the timeout in seconds.

6.1.2 Configure Urls

You should include the scribbler urls in your root url patterns.

```
urlpatterns = patterns('',
    # Other url patterns would go here
    url(r'^scribbler/', include('scribbler.urls')),
)
```

6.1.3 Create Database Tables

You'll need to create the necessary database tables for storing scribble content. This is done with the `syncdb` management command built into Django:

```
python manage.py syncdb
```

django-scribbler uses South to handle database migrations. If you are also using South then you should specify `SOUTH_MIGRATION_MODULES` in settings:

```
SOUTH_MIGRATION_MODULES = {
    'scribbler': 'scribbler.south_migrations',
}
```

To run south migrations call:

```
python manage.py migrate scribbler
```

Note: The latest release of South does not support Python 3. If you want to try django-scribbler with Python 3 you will have go without South for the time being or you should use Django 1.7-1.8 migrations

6.1.4 User Permissions

To edit scribbles on the front-end users must have the `scribbler.add_scribble` and `scribbler.change_scribble` permissions. You can configure uses to have these permissions through the users section of the Django admin. Superusers have all of these permissions by default.

Similarly, to edit fields from models on the front-end, users must have "change" permission for the models being edited. Again these permissions can be configured through the users section of the Django admin.

6.1.5 Include Static Resources

django-scribbler includes both CSS and JS resources which need to be included in your templates to handle the front-end content management. Since you may want to include scribbles on any page on your site these should be included in your base template `<head>`.

```
<link rel="stylesheet" href="{{ STATIC_URL }}scribbler/css/scribbler.css">
<script data-main="{{ STATIC_URL }}scribbler/js/scribbler-min" src="{{ STATIC_URL }}scribbler/libs/re
```

This uses RequireJS to load the additional JS resources. The front-end editor uses CodeMirror (currently using v2.38) which is included in the distribution. Both RequireJS and CodeMirror are available a MIT-style license compatible with this project's BSD license. You can find the license files included in `scribbler/static/scribbler/libs/`.

Note: Prior to v0.5 you also needed to include the `codemirror.css` prior to `scribbler.css`. As of v0.5 you only need to include `scribbler.css`.

Also prior to v0.5 it was recommended to use `{{ STATIC_URL }}scribbler/js/scribbler`. As of v0.5 it is recommended that you use the minified version.

6.1.6 Place Scribbles in Your Template

You are now ready to place the scribble content blocks throughout your templates. This is done with the `scribble` block tag. The basic usage of the tag takes one argument which is the slug name for the scribble. Slugs must be unique per url/slug pair. That means you cannot use the same slug more than once in the template but you can use the same slug in different templates as long as they are rendered on different urls.

```
{% load scribbler_tags %}
{% scribble 'header' %}
    <p>Blip {% now 'Y' %} {{ STATIC_URL|upper }}</p>
{% endscribble %}
```

The content inside the block is the default content that will be rendered if a matching scribble in the database is not found.

The `scribble` tag can take an optional argument which allows for defining shared scribbles.

```
{% load scribbler_tags %}
{% scribble 'header' 'shared' %}
    <p>Blip {% now 'Y' %} {{ STATIC_URL|upper }}</p>
{% endscribble %}
```

The second argument defines a lookup vector to a shared scribble. This overrides the url portion of the url/slug pair, and allows for reuse across multiple templates.

Note: Scribble content can be any valid Django template. However the content does not include all of the context of the template. Only the context provided by the set of `TEMPLATE_CONTEXT_PROCESSORS`.

A second scribbler tag, `scribble_field`, allows for editing fields of model instances. For example, suppose you have a `DaysLog` model with a field named `happenings`. Suppose an instance of this model is passed into your template in the template variable `days_log`. Then the `happenings` field of this `DaysLog` instance can be displayed and edited on the page by including this `scribble_field` template tag in the template for the page:

```
{% load scribbler_tags %}
{% scribble_field days_log 'happenings' %}
```

Note: The logged-in user must have "change" permission for the model in order for the model instance to be editable on the page.

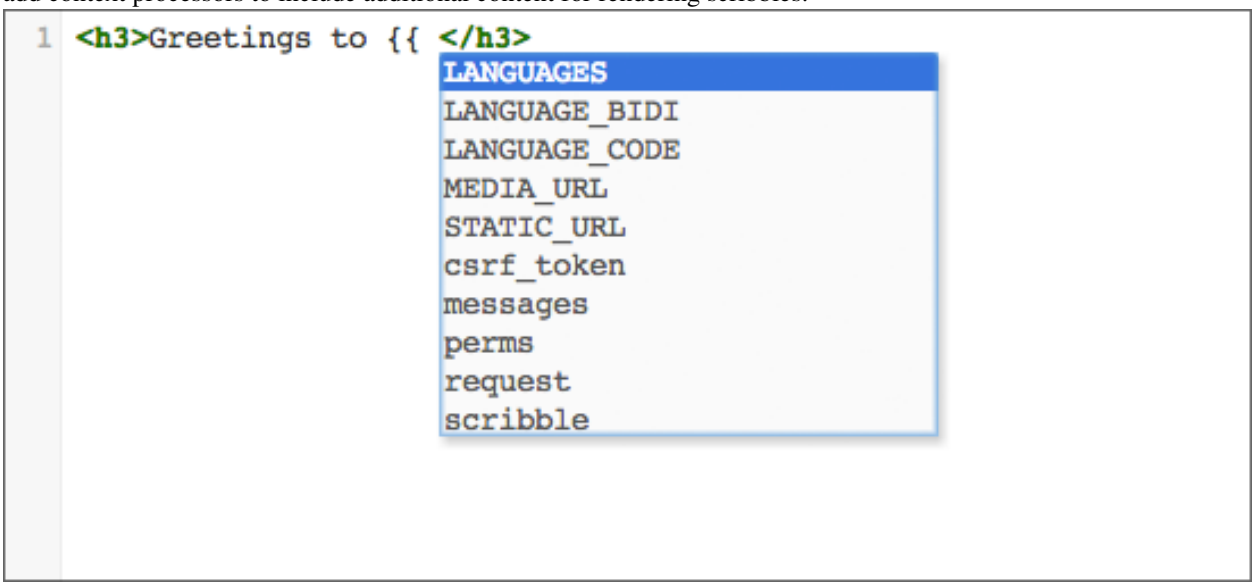
That should be enough to get you up and running with django-scribbler.

6.2 Using the Editor

django-scribbler makes use of CodeMirror to create a powerful client-side editor. We've added a couple features to make it easier when working with Django templates.

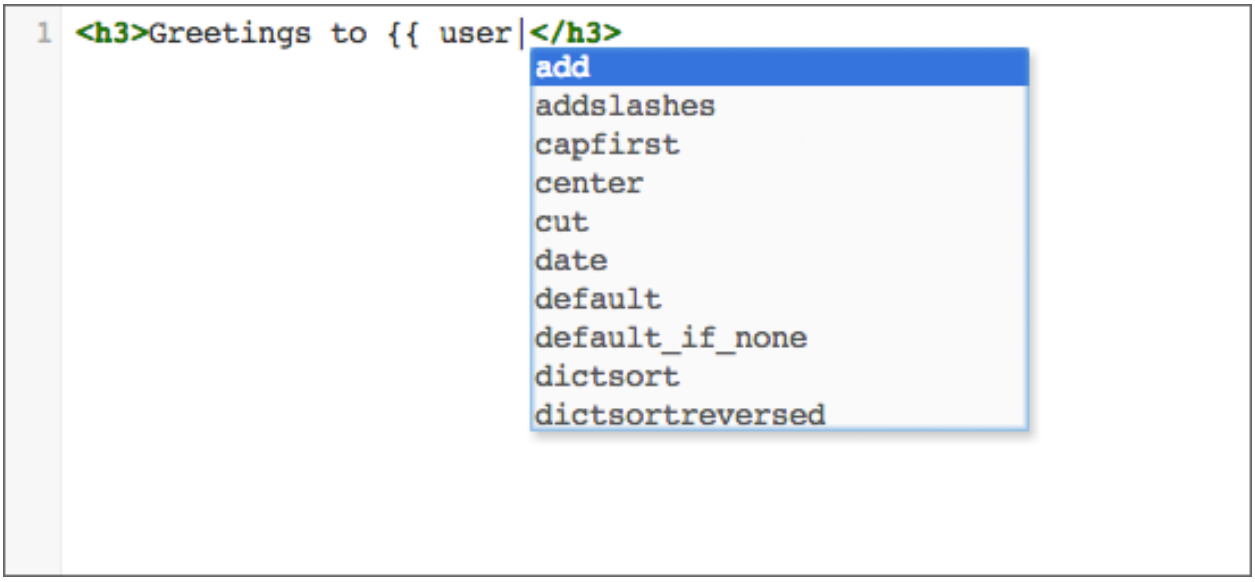
6.2.1 Context Inspection

When using the editor, you can inspect the current context by starting a variable node with `{{` and hitting tab. As noted in the quick start introduction, scribble content can be any valid Django template. The context provided when rendering the scribble includes anything added by the set of `TEMPLATE_CONTEXT_PROCESSORS`. This would include `STATIC_URL`, `MEDIA_URL`, `LANGUAGE_CODE`, the current `user` and others. Developers may choose to add context processors to include additional content for rendering scribbles.



6.2.2 Tempate Tag/Filter Completion

Similar to how the editor can tab-complete the context variables, you can tab complete template tags when `{%` has been opened. The built-in filters can be tab-completed when the pipe `|` character is detected inside of a variable node. Currently this will only complete the built-in tags and filter and will not include any additional tags or filters which might be added by loading additional libraries inside the scribble.



6.2.3 Saving Drafts

While editing the scribble content, the editor will periodically save the current editor content as a draft. These drafts are saved on the client side using local storage (if supported by the browser) or a cookie. While that means you won't be able to see or use these drafts in another browser, it does mean that you work will not be lost if there is a failure on the server while saving changes or making edits. When the editor is opened it will restore any draft that is found. There is the option to discard a working draft. This will load the current scribble content into the editor.

6.3 Writing Editor Plugins

django-scribbler editor has some nice features like a live preview, auto-saving drafts and tab completion of template tags and template context. If you find yourself wanting to extend the functionality of the editor or top menu you can write a plugin.

6.3.1 Basic Setup

Client-side plugins to be added to a `scribbler/js/plugins` folder inside of static files folder. If you are writing this plugin for a reusable application then it would live inside of the app's `static` folder. The plugin name will be the name of the .js file. For example we might create a `demo` plugin by adding a `scribbler/js/plugins/demo.js` inside of our app `static` folder.

6.3.2 Writing the Plugin

django-scribbler uses RequireJS to load its Javascript requirements. Since this code is loaded in a closure, plugins need to be written in the AMD format. The basic format is:

```
define(function () {
    function plugin(editor, menu) {
        // Plugin code goes here...
    }
```

```
    return plugin;
});
```

This has an advantage in that the plugin code can declare requirements as well as take advantage of existing modules used by the scribbler code. For instance if your plugin requires jQuery you can define the requirement:

```
define(['jquery'], function ($) {
    function plugin(editor, menu) {
        // Plugin code goes here...
    }
    return plugin;
});
```

The plugin code itself should return a function which takes two arguments: `editor` and `menu`. Each are the current instance of the editor and the menu respectively. Within the plugin you can add additional controls or event bindings. The APIs for both the editor and the menu are given below.

Note: The plugins are executed after the editor and menu have been initialized but they are loaded asynchronously. That means the editor and menu may be fully rendered before the plugins are executed.

6.3.3 Enabling the Plugin

Plugins are enable by listing them in a `data-scribbler-plugins` attribute on the script tag:

```
<script data-scribbler-plugins="themes"
    data-main="{{ STATIC_URL }}scribbler/js/scribbler{% if not debug %}-min{% endif %}"
    src="{{ STATIC_URL }}scribbler/libs/require.js"></script>
```

If mutliple plugins used then they should be comma seperated as in `data-scribbler-plugins="themes,other"`.

Note: Since the plugins are loaded asynchronously they might not load in the same order they are listed. Plugins should be written and designed with that limitation in mind.

6.3.4 Available Libraries

As noted above you can use the `define` call to load additional dependencies/libraries for your plugin from the set of libraries used by django-scribbler. The available libraries are:

- require: RequireJS 2.1.4
- jquery: jQuery 1.8.3
- codemirror: CodeMirror 2.38
- underscore: Underscore 1.4.4
- backbone: Backbone 0.9.10

6.3.5 Editor API

The `editor` passed in the plugin is an instance of the `ScribbleEditor` defined in `scribbler/js/scribbler-editor.js`. Below is a list of some of the most relevant functions and properties for controlling the editor.

`editor.`**`scribbles`**
: This is a jQuery object containing all of the scribble divs available on the page for editting.

`editor.`**`footerControls`**
: A jQuery object for the div wrapping all of the editor button controls (Save, Save Draft, Discard Draft, Close). If you want to add additional controls they should be appended here.

`editor.`**`editor`**
: `editor.editor` is the instance of the CodeMirror editor. You can manipulate this object to change the options with `editor.editor.setOption`. See the CodeMirror usage manual for available options: http://codemirror.net/doc/manual.html

`editor.`**`open`**(*scribble*)
: Opens the editor to edit the given scribble.

`editor.`**`close`**()
: Closes the editor.

`editor.`**`submitPreview`**(*force*)
: Submits the current editor content to render the live preview. By default this will not submit if the editor is currently in the process of rendering a preview. Passing `true` into the call will force the submission.

`editor.`**`submitSave`**()
: Submits the editor content to save the current scribble content. By default the save will not be submitted if the last preview was not valid.

`editor.`**`getFormData`**()
: Prepares the current form data for preview/save submission. If you want to pass additional data to the server your plugin could extend this function.

`editor.`**`createDraft`**()
: Saves the current editor content as a local draft.

`editor.`**`restoreDraft`**()
: Restores the editor content from the last saved draft if available.

`editor.`**`deleteDraft`**()
: Discards last saved draft.

`editor.`**`setStatus`**(*msg*)
: Displays a status message to the user in the header of the editor.

`editor.`**`destroy`**()
: Removes the editor from the DOM and unbinds all event handling.

6.3.6 Menu API

The `menu` passed in the plugin is an instance of the `ScribbleMenu` defined in `scribbler/js/scribbler-menu.js`. Below is a list of some of the most relevant functions and properties for controlling the menu.

`menu.`**`scribbles`**
: This is a jQuery object containing all of the scribble divs available on the page for editing.

`menu.`**`menuControls`**
: A jQuery object for the div wrapping all of the menu button controls. If you want to add additional controls they should be appended here.

`menu.`**`open`**()
: Opens the top menu bar.

`menu.close()`
Closes the top menu bar.

`menu.toggle()`
Toggles the open/close state of the top menu bar.

`menu.highlight()`
Highlights all editable scribble areas on the page.

`menu.destroy()`
Removes the menu from the DOM and unbinds all event handling.

6.4 Contributing Guide

There are a number of ways to contribute to django-scribbler. If you are interested in making django-scribbler better then this guide will help you find a way to contribute.

6.4.1 Ways to Contribute

You can contribute to the project by submitting bug reports, feature requests or documentation updates through the Github issues.

6.4.2 Translate django-scribbler

We are working towards translating django-scribbler into different languages. There are only a few strings to translate so it is a great way to be involved with the project. The translations are managed through Transifex. Please do not submit translate requests/updates to the Github repo.

6.4.3 Getting the Source

You can clone the repository from Github:

```
git clone git://github.com/caktus/django-scribbler.git
```

However this checkout will be read only. If you want to contribute code you should create a fork and clone your fork. You can then add the main repository as a remote:

```
git clone git@github.com:<your-username>/django-scribbler.git
cd django-scribbler
git remote add upstream git://github.com/caktus/django-scribbler.git
git fetch upstream
```

django-scribbler requires a few static libraries which are not included in the repository. Before beginning development you should make sure you have these libraries with:

```
make fetch-static-libs
```

6.4.4 Running the Tests

When making changes to the code, either fixing bugs or adding features, you'll want to run the tests to ensure that you have not broken any of the existing functionality. With the code checked out and Django installed you can run the tests via:

```
python setup.py test
```

or:

```
python runtests.py
```

Note that the tests require the mock library. To test against multiple versions of Django you can use install and use `tox>=1.4`. The `tox` command will run the tests against Django 1.3, 1.4 and the current Git master using Python 2.6.:

```
# Build all environments
tox
# Build a single environment
tox -e py26-1.3.X
```

Building all environments will also build the documentation. More on that in the next section.

The JS plugins are tested using the *QUnit <http://qunitjs.com/>* testing framework. You can run the tests by opening `scribbler\tests\qunit\index.html` in your browser. You can also run the tests using the *PhantomJS <http://phantomjs.org/>* headless runner. First install PhantomJS from NPM (requires at least 1.6):

```
# Install phantomjs from the NPM package
npm install phantomjs -g
# Run QUnit tests
phantomjs scribbler/tests/qunit/runner.js scribbler/tests/qunit/index.html
```

We've added a make command which you can use as well:

```
make test-js
```

6.4.5 Building the Documentation

This project aims to have a minimal core with hooks for customization. That makes documentation an important part of the project. Useful examples and notes on common use cases are a great way to contribute and improve the documentation.

The docs are written in ReST and built using Sphinx. As noted above you can use tox to build the documentation or you can build them on their own via:

```
tox -e docs
```

or:

```
make html
```

from inside the `docs/` directory.

6.4.6 Building the CSS

The CSS used by django-scribbler is built using LESS. No changes should be made to the `scribbler.css` directly. Instead changes should be made to the `scribbler.less` file. After changes are made to `scribbler.less` you can create the new compressed CSS with the Node based complier. In addition this uses the RequireJS optimizer to inline the required `codemirror.css`:

```
# Install less and requirejs from the NPM package
npm install less requirejs -g
make build-css
```

The example project uses the client-side LESS compiler to make local development easier.

6.4.7 Building the JS

While it is not often needed for local development, the final released JS is bundled and minified using the same RequireJS optimizer used for the CSS. To build `scribbler-min.js` you should have the optimizer installed and run:

```
make build-js
```

6.4.8 Coding Standards

Code contributions should follow the PEP8 and Django contributing style standards. Please note that these are only guidelines. Overall code consistency and readability are more important than strict adherence to these guides.

The Javascript is configured for some basic JSHint checks. Changes to the Javascript should pass without errors. You can check the Javascript file on the command line with Node based CLI tool:

```
# Install jshint from the NPM package
npm install jshint -g
# Check the scribbler JS
jshint scribbler/static/scribbler/js/
```

This can also be done with the `make` command:

```
make lint-js
```

6.4.9 Submitting a Pull Request

The easiest way to contribute code or documentation changes is through a pull request. For information on submitting a pull request you can read the Github help page https://help.github.com/articles/using-pull-requests.

Pull requests are a place for the code to be reviewed before it is merged. This review will go over the coding style as well as if it solves the problem intended and fits in the scope of the project. It may be a long discussion or it might just be a simple thank you.

Not necessarily every request will be merged but you should not take it personally if you change is not accepted. If you want to increase the chances of your change being incorporated then here are some tips.

- Address a known issue. Preference is given to a request that fixes a currently open issue.
- Include documentation and tests when appropriate. New features should be tested and documented. Bugfixes should include tests which demonstrate the problem.
- Keep it simple. It's difficult to review a large block of code so try to keep the scope of the change small.

You should also feel free to ask for help writing tests or writing documentation if you aren't sure how to go about it.

6.4.10 Installing an Unstable Release

Since the built CSS, JS and other static dependencies are not included in the repository, it is not possible to install django-scribbler directly from Github. If you want to install and unstable version of django-scribbler you have a few options.

Warning: While we try to keep the `master` branch stable, there may be bugs or unfinished work there. It is recommended that you use a stable release of django-scribbler when possible.

Install Local Build

The step overview for installing from a local build is:

- Check out the repository
- Install static libraries
- Build CSS and JS
- Install from local repository

From the command line this would be:

```
git clone git://github.com/caktus/django-scribbler.git
cd django-scribbler
make fetch-static-libs build-css build-js
pip install .
```

Create an Unstable Package

Installing from a local build is probably a reasonable solution for a single person wanting to test out the current master or a feature branch in a large project. However, it isn't a good solution if you want to deploy this to a larger testing environment or multiple computers. The basic steps are more or less the same:

- Check out the repository
- Install static libraries
- Build CSS and JS
- Create a source distribution
- Distribute .tar file
- Install for packaged .tar

From the command line this would be:

```
git clone git://github.com/caktus/django-scribbler.git
cd django-scribbler
make fetch-static-libs build-css build-js
python setup.py sdist
```

This will create a `django-scribbler-X.X.X.tar.gz` inside a `dist/` directory where `X.X.X` is the current `scribbler.__version__`. This tar file would then be distributed using your favorite file hosting service (S3, Dropbox, etc). You can then install by using `pip`:

```
pip install http://path-to-hostedfile/django-scribbler-X.X.X.tar.gz
```

6.5 Release History

Release and change history for django-scribbler

6.5.1 v0.6.0 (Released 2015-10-7)

This release fixes some lingering issues with Django 1.8 support and integrates Wheel support and features from the latest versions of Tox.

Features

- Added Wheel support (#96)
- Updated Tox and Travis to work with Tox 2.0+ (#90)
- Changed button color on editor
- Confirmed Python 3.4 support

Bug Fixes

- Fixed issues with Django 1.8 compatibility (#84)

6.5.2 v0.5.3 (Released 2014-06-13)

- Fixed issues with Python 3 compatibility
- Fixed issues with Django 1.7+ compatibility
- Fixed issues with IE 8 compatibility

6.5.3 v0.5.2 (Released 2013-05-02)

- Fixed issue with scribbler styles overlapping/overriding site styles. See #73

6.5.4 v0.5.1 (Released 2013-03-03)

- Bug fix release for broken packaging

6.5.5 v0.5.0 (Released 2013-03-03)

This release includes a major refactor of the JS code and adds support for writing client-side plugins for customizing the editor.

Features

- Upgraded to CodeMirror 3.02
- Additional build/development utilities and documentation
- Started including a minified and optimized version of scribbler.js for production usage
- CSS is now built to include the base CodeMirror CSS and does not need to be added to the template separately

Bug Fixes

- Fixed a bug where you could not follow an internal link in the scribble content. See #66

Backwards Incompatible Changes

The static dependencies (RequireJS, CodeMirror and jQuery) were originally included in the repository but have been removed. These are still included in the final distribution. However, if you installing django-scribbler directly from git these will no longer be available. See the contributing guide for more information on building/installing an unstable version.

6.5.6 v0.4.0 (Released 2013-01-01)

The length of the slug field has been reduced to fix problems with the unique contraint on MySQL. Upgrading requires running a migration:

```
manage.py migrate scribbler
```

Features

- Top level menu to reveal all editable sections on the page
- i18n support and initial French translation thanks to Nicolas Ippolito
- Created Transifex group for translations
- Added optional parameter to scribble tag to support shared scribbles thanks to David Ray
- Added the ability to discard a saved draft

Bug Fixes

- Fixed bug with newly included jQuery overriding an existing version. See #53
- Fixed bug with unique index on MySQL thanks to David Ray. See #61

Backwards Incompatible Changes

- The fix for #61 reduced the length of the slug field from 255 characters to 64

6.5.7 v0.3.0 (Released 2012-10-26)

Features

- Autocomplete for Django template tags and filters
- New scribble_field template tag to allow editing of fields in arbitrary models

6.5.8 v0.2.1 (Released 2012-10-12)

Bug Fixes

- Preview was broken when scribble was saved due to unique constraint. See #34

6.5.9 v0.2.0 (Released 2012-10-12)

The editor now saves drafts on the client side by default. Python 3 support is added when using the lastest Django master. There is also some additional documentation.

A unique constraint was added and upgrading from v0.1 does require a migration:

```
manage.py migrate scribbler
```

- Added experimental Python >= 3.2 support when using Django 1.5dev
- Caktus Consulting Group has taken over the primary development
- Added the ability to save as a draft on the client side
- Added an official contributing guide

Bug Fixes

- Added unique constraint for url/slug pair. South migration is included.

6.5.10 v0.1.1 (Released 2012-08-25)

Minor bug fix release for some JS and CSS issues.

Bug Fixes

- Fixed issue with the content editor z-index allowing content in front when open
- Fixed issue where links within editable content could not be clicked by editors

6.5.11 v0.1.0 (Released 2012-07-28)

- Initial public release.

Features

- Template tag for rendering content blocks
- CodeMirror editor integration

E

M