django-precise-bbcode Documentation

Release 1.0.x

Morgan Aubert

Apr 04, 2017

Contents

Django-precise-bbcode is a Django application providing a way to create textual contents based on BBCodes.

> BBCode is a special implementation of HTML. BBCode itself is similar in style to HTML, tags are enclosed in square brackets [and] rather than < and > and it offers greater control over what and how something is displayed.

This application includes a BBCode compiler aimed to render any BBCode content to HTML and allows the use of BBCodes tags in models, forms and admin forms. The BBCode parser comes with built-in tags (the default ones ; `b`, `u`, `i`, etc) and allows the definition of custom BBCode tags, placeholders and smilies.

CHAPTER 1

Features

- BBCode parser for rendering any string containing bbcodes
- Tools for handling bbcode contents: templatetags, model field
- Support for custom bbcode tags:
  - Simple custom bbcodes can be defined in the Django admin
  - ... or they can be registered using a plugin system by defining some Python classes
- Support for custom bbcode placeholders
- Support for custom smilies and emoticons

CHAPTER 2

Using django-precise-bbcode

Getting started

Requirements

- Python 2.7, 3.3, 3.4 or 3.5
- Django 1.8.x, 1.9.x, 1.10.x or 1.11.x
- Pillow 2.2. or higher

Installing

Install *django-precise-bbcode* using:

```
pip install django-precise-bbcode
```

Add `precise_bbcode` to `INSTALLED_APPS` in your project's settings module:

```
INSTALLED_APPS = (
    # other apps
    'precise_bbcode',
)
```

Then install the models:

```
python manage.py migrate
```

Basic reference & guidance

Once you have *django-precise-bbcode* working, you'll want to explore its features and capabilities in more depth. Here you will learn to use the BBCodes provided by the module.

Built-in BBCode tags

Django-precise-bbcode comes with some built-in BBCode tags that you can use to render any content based on bbcodes. The built-in bbcodes are as follows:

BBCode	Function	Options	Examples
`b`	creates bold text		[b]bold text[/b]
`i`	creates italic text		[i]italice text[/i]
`u`	creates underlined text		[u]underlined text[/u]
`s`	creates striked text		[s]striked text[/s]
`list`	creates an unordered list	1: ordered list 01: ordered list (leading zero) a: ordered list (lower alpha) A: ordered list (upper alpha) i: ordered list (lower roman) I: ordered list (upper roman)	[list][*]one[*]two[/list] [list=1][*]one[*]two[/list]
`*`	creates a list item		[list][*]one[*]two[/list]
`code`	retains all formatting		[code][b]example[/b][/code]
`quote`	creates a blockquote		[quote]quoted string[/quote]
`center`	creates a centered block		[center]example[/center]
`color`	changes the colour of a text		[color=red]red text[/color] [color=#FFFFFF]white text[/color]
`url`	creates a URL		[url]http://example.com[/url] [url=http://example.com]text[/url]
`img`	displays an image		[img]http://xyz.com/logo.png[/img]

Rendering BBCodes

BBcode parser

Django-precise-bbcode provides a BBCode parser that allows you to transform any textual content containing BBCode tags to the corresponding HTML markup. To do this, just import the `get_parser` shortcut and use the `render` method of the BBCode parser:

```
from precise_bbcode.bbcode import get_parser

parser = get_parser()
rendered = parser.render('[b]Hello [u]world![/u][/b]')
```

This will store the following HTML into the `rendered` variable:

```
<strong>Hello <u>world!</u></strong>
```

Template tags

The previous parser can also be used in your templates as a template filter or as a template tag after loading `bbcode_tags`:

```
{% load bbcode_tags %}
{% bbcode entry.bbcode_content %}
{{ "[b]Write some bbcodes![/b]"|bbcode }}
```

Doing this will force the BBCode content included in the `entry.bbcode_content` field to be converted to HTML. The last statement will output:

```
<strong>Write some bbcodes!</strong>
```

Jinja2 template support

Django-precise-bbcode supports Jinja 2 templating. You have to add the `precise_bbcode.jinja2tags.bbcode` extension to your template extensions if you want to use the *django-precise-bbcode*'s Jinja 2 tags in your project:

```
TEMPLATES = [
    # ...
    {
        'BACKEND': 'django.template.backends.jinja2.Jinja2',
        'APP_DIRS': True,
        'OPTIONS': {
            'extensions': [
                'precise_bbcode.jinja2tags.bbcode',
            ],
        },
    }
]
```

The BBCode parser can then be used into your Jinja 2 templates as a function or as a template filter:

```
{{ bbcode("[b]Write some bbcodes![/b]") }}
{{ "[b]Write some bbcodes![/b]"|bbcode }}
```

Storing BBCodes

The Django built-in `models.TextField` is all you need to simply add BBCode contents to your models. However, a common need is to store both the BBCode content and the corresponding HTML markup in the database. To address this *django-precise-bbcode* provides a `BBCodeTextField`:

```
from django.db import models
from precise_bbcode.fields import BBCodeTextField

class Post(models.Model):
    content = BBCodeTextField()
```

A `BBCodeTextField` field contributes two columns to the model instead of a standard single column : one is used to save the BBCode content ; the other one keeps the corresponding HTML markup. The HTML content of such a field can then be displayed in any template by using its `rendered` attribute:

```
{{ post.content.rendered }}
```

Note: Please note that you should to use the `safe` template filter when displaying your BBCodes if you choose to store BBCode contents in the built-in `models.TextField`. This is because `models.TextField` contents are escaped by default.

```
{% load bbcode_tags %}
{{ object.bbcode_content|safe|bbcode }}
```

Extending Precise BBCode

Django-precise-bbcode allows the creation of custom BBCode tags, placeholders and smilies.

Custom BBCode tags

While *django-precise-bbcode* comes with some built-in BBCode tags, there will be times when you need to add your own.

Defining BBCode tags through the admin site

The easy way.

Django-precise-bbcode provides a `BBCodeTag` model which can be seen as a helper to allow end users to easily define BBCode tags. Just go to the admin page and you will see a new 'BBCodes' section. In this section you can create and edit custom BBCode tags. These are then used by the built-in BBCode parser to render any BBCode content.

Adding a custom BBCode tag consists in defining at least two values in the associated admin form, both its usage (tag definition) and its HTML replacement code.

Tag definition

The tag definition is the expression of the bbcode usage. It's where you enter your bbcode. All you need to do is to add a string containing your BBCode and the associated placeholders (special uppercase words surrounded by { and } – they are similar to the "replacement fields" that you define in Python format strings):

```
[foo]{TEXT}[/foo]
```

In this example, we have a bbcode named `foo` which can contain some text (`{TEXT}` placeholder).

So a bbcode definition takes the form of what users will enter when using this bbcode, except that all parts of the bbcode where data is required are expressed as placeholders. The placeholders that you can use in such a tag definition are typed. This means that some semantic verifications are done before rendering in order to ensure that data containing non-allowed characters are not converted to HTML.

Django-precise-bbcode provides some placeholders by default, such as `{TEXT}`, `{SIMPLETEXT}`, `{COLOR}`, `{NUMBER}`, `{URL}` or `{EMAIL}`. For a full list of the available placeholders and the techniques used to define custom placeholders, please refer to *Custom BBCode placeholders*.

Note that you can specify an option to your bbcode in its definition:

```
[foo={COLOR}]{TEXT}[/foo]
```

In this case, the data associated with the `{COLOR}` placeholder is not required at runtime. If you wish to use two placeholders of the same type in your bbcode definition, you have to append a number to their respective names (eg. `{TEXT1}`):

```
[foo={TEXT1}]{TEXT2}[/foo]
```

HTML replacement code

The HTML replacement code is where you enter the HTML for the bbcode you defined previously. All the placeholders you used in your bbcode definition must be present in the HTML replacement code. For example, the HTML replacement code associated with the last `[foo]` bbcode example could be:

```
<div style="background:{COLOR};">{TEXT}</div>
```

BBCode options

Some specific options can be used when defining a custom bbcode to alter its default behavior. For example, you could want to forbid the rendering of any bbcode tags included inside your new bbcode. All these options are boolean fields and are indicated in the following table:

Option	Definition	Default
newline_closes	Force the closing of a tag after a newline	False
same_tag_closes	Force the closing of a tag after the beginning of a similar tag	False
end_tag_closes	Force the closing of a tag after the end of another tag	False
standalone	Set this option if a tag does not have a closing tag (eg. [hr])	False
transform_newlines	Convert any line break to the equivalent markup	True
render_embedded	Force the tags embedded in a tag to be rendered	True
escape_html	Escape HTML characters (<, >, and &) inside a tag	True
replace_links	Replace URLs with link markups inside a tag	True
strip	Strip leading and trailing whitespace inside a tag	False
swallow_trailing_newline	Swallow the first trailing newline inside a tag	False

Defining BBCode tags plugins

The fun part.

While the previous bbcode tag system allows you to easily define various bbcodes, you may want to do more complex treatments with your bbcodes (eg. handle other types of data). You may also want to write some **reusable** or **generic** bbcode tags.

To do so, you will have to write a subclass of `precise_bbcode.bbcode.tag.BBCodeTag` for any tag you want to create. These class-based bbcode tags must be defined inside a `bbcode_tags` Python module in your Django application (just add a file called `bbcode_tags.py` to an existing Django application). And last, but not least, your class-based bbcode tags must be registered to the `precise_bbcode.tag_pool.tag_pool` object by using its `register_tag` method to be available to the BBCode parser.

Each of these tags must provide a `name` attribute and can operate in two different ways:

- The `BBCodeTag` subclass provides a `definition_string` attribute and a `format_string` attribute. In this case, the tag will operate as previously described. The `definition_string` corresponds to the tag definition and defines how the tag should be used. The `format_string` is the HTML replacement code that will be used to generate the final HTML output
- The `BBCodeTag` subclass implements a `render` method that will be used to transform the bbcode tag and its context (value, option if provided) to the corresponding HTML output

Defining bbcodes based on a definition string and a format string

In this case, you have to provide a `definition_string` attribute and a `format_string` attribute to your `BBCodeTag` subclass, in addition to the `name` attribute.

Let's write a simple example. Consider we are trying to write a `bar` bbcode which will strike any text placed inside its tags. So we could write:

```
# bbcode_tags.py
from precise_bbcode.bbcode.tag import BBCodeTag
from precise_bbcode.tag_pool import tag_pool

class BarBBCodeTag(BBCodeTag):
    name = 'bar'
    definition_string = '[bar]{TEXT}[/bar]'
    format_string = '<strike>{TEXT}</strike>'

tag_pool.register_tag(BarBBCodeTag)
```

Note that you can use any BBCode options specified previously to alter the default behavior of your class-based tags (see *BBCode options*). To do so, give your bbcode tag options by using an inner class `Options`, like so:

```
# bbcode_tags.py
from precise_bbcode.bbcode.tag import BBCodeTag
from precise_bbcode.tag_pool import tag_pool

class BarBBCodeTag(BBCodeTag):
    name = 'bar'
    definition_string = '[bar]{TEXT}[/bar]'
    format_string = '<strike>{TEXT}</strike>'

    class Options:
        render_embedded = False
        strip = False

tag_pool.register_tag(BarBBCodeTag)
```

Defining bbcodes based on a `render` method

In this case, each of your `BBCodeTag` subclasses must provide a `name` attribute and must implement a `render` method. The `render` method is used to transform your bbcode tag and its context (value, option if provided) to the corresponding HTML output. The `render` method takes three arguments:

- **value**: the context between the start end the end tags, or None for standalone tags. Whether this has been rendered depends on the `render_embedded` tag option

- **option**: The value of an option passed to the tag ; defaults to None
- **parent**: The options (instance of `precise_bbcode.bbcode.tag.BBCodeTagOptions`) associated with the parent bbcode if the tag is being rendered inside another tag, otherwise None

Keep in mind that your `render` method may have to validate the data associated with your tag before rendering it. Any validation process should be triggered from this `render` method.

Let's write another example. Consider we are trying to write a `rounded` bbcode which will surround inside a rounded frame any text placed inside the tags. If provided, the option passed to the tag is assumed to be a colour in order to modify the resulting HTML code. So we could write:

```
# bbcode_tags.py
import re
from precise_bbcode.bbcode.tag import BBCodeTag
from precise_bbcode.tag_pool import tag_pool

color_re = re.compile(r'^([a-z]+|#[0-9abcdefABCDEF]{3,6})$')

class RoundedBBCodeTag(BBCodeTag):
    name = 'rounded'

    class Options:
        strip = False

    def render(self, value, option=None, parent=None):
        if option and re.search(color_re, option) is not None:
            return '<div class="rounded" style="border-color:{};">{}</div>'.
↪format(option, value)
        return '<div class="rounded">{}</div>'.format(value)

tag_pool.register_tag(RoundedBBCodeTag)
```

Again, you can use any BBCode options as previously stated (see *BBCode options*).

Overriding default BBCode tags

When loaded, the parser provided by *django-precise-bbcode* provides some default bbcode tags (please refer to *Built-in BBCode tags* for the full list of default tags). These default tags can be overriden. You just have to create another tag with the same name either by defining it in the admin site or by defining it in a `bbcode_tags` Python module as previously explained.

Custom BBCode placeholders

When you define bbcode tags, you can choose to use placeholders in order to define where data is required. These placeholders, such as `{TEXT}` or `{NUMBER}` are typed. This means that some semantic verifications are done before rendering in order to ensure that the content corresponding to a specific placeholder is valid. *Django-precise-bbcode* comes with some built-in BBCode placeholders that you can use in your bbcode tag definitions. You can also choose to create your owns.

Built-in placeholders

Placeholder	Usage	Definition
`{TEXT}`		matches anything
`{SIMPLETEXT}`		matches latin characters, numbers, spaces, commas, dots, minus, plus, hyphen and underscore
`{COLOR}`		matches a colour (eg. `red` or `#000FFF`)
`{NUMBER}`		matches a number
`{URL}`		matches a valid URL
`{EMAIL}`		matches a valid e-mail address
`{RANGE}`	`{RANGE=min,max}`	matches a valid number between *min* and *max*
`{CHOICE}`	`{CHOICE=foo,bar}`	matches all strings separated with commas in the placeholder

Defining BBCode placeholders plugins

Django-precise-bbcode allows you to define your own placeholders.

To do so, you will have to write a subclass of `precise_bbcode.bbcode.placeholder.BBCodePlaceholder` for any placeholder you want to create. These class-based bbcode placeholders must be defined inside a `bbcode_placeholders` Python module in your Django application (just add a file called `bbcode_placeholders.py` to an existing Django application). In the same way as bbcode tag classes, your class-based bbcode placeholders must be registered to the `precise_bbcode.placeholder_pool.placeholder_pool` object by using its `register_placeholder` method to be available to the BBCode parser.

Each bbcode placeholder must have a `name` attribute and can operate in two different ways:

- The `BBCodePlaceholder` subclass provides a `pattern` attribute, which is a valid regular expression. In this case, a given content will be valid in the context of the placeholder if it matches this regular expression
- The `BBCodePlaceholder` subclass implements a `validate` method. This method is used to check whether a given content is valid according to the placeholder definition associated to it and should return `True` or `False`

Defining placeholders based on a regular expression pattern

In this case, you have to provide a `pattern` attribute to your `BBCodePlaceholder` subclass, in addition to the `name` attribute.

Let's write a simple example. Consider we are trying to write a `{PHONENUMBER}` bbcode placeholder which will allow end-users to fill some bbcode tags with phone numbers. So we could write:

```
# bbcode_placeholders.py
import re
from precise_bbcode.bbcode.placeholder import BBCodePlaceholder
from precise_bbcode.placeholder_pool import placeholder_pool

class PhoneNumberBBCodePlaceholder(BBCodePlaceholder):
    name = 'phonenumber'
    pattern = re.compile(r'(\d{3}[-\.\s]??\d{3}[-\.\s]??\d{4}|\(\d{3}\)\s*\d{3}[-\.
↪\s]??\d{4}|\d{3}[ \.\s]??\d{4})')
```

```
placeholder_pool.register_placeholder(PhoneNumberBBCodePlaceholder)
```

So if this placeholder is used inside, let's say, a `[telto]` bbcode tag, `+33000000000` will be a valid input.

Defining placeholders based on a `validate` method

In this case, each of your `BBCodePlaceholder` subclasses must provide a `name` attribute and must implement a `validate` method. This method is used to check whether the input associated with a placeholder is valid and can be rendered. The `validate` method takes two arguments:

- **content**: the content used to fill the placeholder that must be validated
- **extra_context**: the extra context of the placeholder if defined in a tag definition

Note that the extra context is a string defined in the placeholder in a bbcode tag definition. For example, `CHOICE` is the placeholder name and `apple,tomato` is the extra context if the `CHOICE` placeholder is used as follows inside a bbcode tag definition: `{CHOICE=apple,tomato}`.

Let's write an example. Consider we are trying to write a `{RANGE}` placeholder which will allow end-users to fill some bbcode tags with a number that will be valid only if it is within a specific range. So we could write:

```
# bbcode_placeholders.py
import re
from precise_bbcode.bbcode.placeholder import BBCodePlaceholder
from precise_bbcode.placeholder_pool import placeholder_pool

class RangeBBCodePlaceholder(BBCodePlaceholder):
    name = 'range'

    def validate(self, content, extra_context):
        try:
            value = float(content)
        except ValueError:
            return False

        try:
            min_content, max_content = extra_context.split(',')
            min_value, max_value = float(min_content), float(max_content)
        except ValueError:
            return False

        if not (value >= min_value and value <= max_value):
            return False

    return True


placeholder_pool.register_placeholder(RangeBBCodePlaceholder)
```

The `validate` method allows you to implement your own validation logic for your custom placeholders.

Overriding default BBCode placeholders

When loaded, the parser provided by *django-precise-bbcode* provides some default bbcode placeholders (please refer to *Built-in placeholders* for the full list of default placeholders). These default placeholders can be overridden. You

just have to register another placeholder with the same name and it will override the default one.

Custom smilies

Django-precise-bbcode does not come with some built-in smilies but supports adding custom smilies and emoticons through the Django admin.

To add a smiley, just go to the admin page and you will see a new 'Smileys' section. In this section you can create and edit custom smilies. The smileys defined through the Django admin are then used by the built-in BBCode parser to transform any *smiley code* to the corresponding HTML.

Adding a smiley consists in filling at least the following fields:

- `code`: The smiley code - it's the textual shortcut that the end users will use to include emoticons inside their bbcode contents. This text can be composed of any character without whitespace characters (eg. `;)` or `-_-`)
- `image`: The smiley image

The size at which the emoticon image is rendered can also be specified by using the `image_width` and `image_height` fields.

Settings

This is a comprehensive list of all the settings *Django-precise-bbcode* provides. All settings are optional.

Parser settings

BBCODE_NEWLINE

Default: `'<br />'`

The HTML replacement code for a default newline.

BBCODE_ESCAPE_HTML

Default:

```
(
    ('&', '&amp;'),
    ('<', '&lt;'),
    ('>', '&gt;'),
    ('"', '&quot;'),
    ('\'', '&#39;'),
)
```

The list of all characters that must be escaped before rendering.

BBCODE_DISABLE_BUILTIN_TAGS

Default: `False`

The flag indicating whether the built-in BBCode tags should be disabled or not.

BBCODE_ALLOW_CUSTOM_TAGS

Default: `True`

The flag indicating whether the custom BBCode tags (those defined by the end users through the Django admin) are allowed.

BBCODE_NORMALIZE_NEWLINES

Default: `True`

The flag indicating whether the newlines are normalized (if this is the case all newlines are replaced with `\r\n`).

Smilies settings

BBCODE_ALLOW_SMILIES

Default: `True`

The flag indicating whether the smilies (defined by the end users through the Django admin) are allowed.

SMILIES_UPLOAD_TO

Default: `'precise_bbcode/smilies'`

The media subdirectory where the smilies should be uploaded.

Indices and tables

- genindex
- modindex
- search