bbcode Documentation

Release 1.0.16

Dan Watson

Sep 27, 2017

Contents

Contents:

CHAPTER 1

Basic Usage

If you need only the *built-in tags*, you can simply use the global default parser:

```
import bbcode
html = bbcode.render_html(text)
```

Basic formatters can be added using simple string substitution. For instance, adding a [wiki] tag for wikipedia links may look like:

```
parser = bbcode.Parser()
parser.add_simple_formatter('wiki', '<a href="http://wikipedia.org/wiki/%(value)s">
↪%(value)s</a>')
```

Custom Parser Objects

The bbcode `Parser` class takes several options when creating:

newline (default: `'<br />'`) What to replace newlines with.

install_defaults (default: `True`) Whether to install the default tag formatters. If False, you will need to specify add tag formatters yourself.

escape_html (default: `True`) Whether to escape special HTML characters (<, >, &, ", and '). Replacements are specified as tuples in `Parser.REPLACE_ESCAPE`.

replace_links (default: `True`) Whether to automatically create HTML links for URLs in the source text.

replace_cosmetic (default: `True`) Whether to perform cosmetic replacements for —, –, ..., (c), (reg), and (tm). Replacements are specified as tuples in `Parser.REPLACE_COSMETIC`.

tag_opener (default: `'['`) The opening tag character(s).

tag_closer (default: `']'`) The closing tag character(s).

linker (default: `None` (use the built-in link replacement)) A function that takes a regular expression match object (and optionally the `Parser` context) and returns an HTML replacement string.

linker_takes_context (default: `False`) Whether the linker function accepts a second `context` parameter. If `True`, the linker function will be passed the context sent to `Parser.format`.

drop_unrecognized (default: `False`) Whether to drop unrecognized (but valid) tags. The default is to leave the tags, unformatted, in the output.

Customizing the Linker

The linker is a function that gets called to replace URLs with markup. It takes one or two arguments (depending on whether you set `linker_takes_context`), and might look like this:

```
def my_linker(url):
    href = url
    if '://' not in href:
        href = 'http://' + href
    return '<a href="%s">%s</a>' % (href, url)

parser = bbcode.Parser(linker=my_linker)
parser.format('www.apple.com') # returns <a href="http://www.apple.com">www.apple.com
↪</a>
```

For an example of a linker that may want the render context, imagine a linker that routes all clicks through a local URL:

```
def my_linker(url, context):
    href = url
    if '://' not in href:
        href = 'http://' + href
    redir_url = context['request'].build_absolute_url('/redirect/') + '?to=' + urllib.
↪quote(href, safe='/')
    return '<a href="%s">%s</a>' % (redir_url, url)

parser = bbcode.Parser(linker=my_linker, linker_takes_context=True)
parser.format('www.apple.com', request=request)
```

CHAPTER 2

Built-In Tags

Below are the tag formatters that are built into `bbcode` by default:

Tag	Input	Output
b	[b]test[/b]	<b>test</b>
i	[i]test[/i]	<i>test</i>
u	[u]test[/u]	<u>test</u>
s	[s]strike[/s]	<span style=”text-decoration:line-through;”>strike</span>
hr	[hr]	<hr />
sub	x [sub]3[/sub]	x<sub>3</sub>
sup	x [sup]3[/sup]	x<sup>3</sup>
list/*	[list] [*]one [*]two [/list]	<ul> <li>one</li> <li>two</li> </ul>
quote	[quote]hello[/quote]	<blockquote>hello</blockquote>
code	[code]x = 3[/code]	<code>x = 3</code>
center	[center]hello[/center]	<div style=”text-align:center;”>hello</div>
color	[color=red]red[/color]	<span style=”color:red;”>red</span>
url	[url=www.apple.com]Apple[/url]	<a href=”http://www.apple.com“>Apple</a>

CHAPTER 3

Advanced Tag Formatters

Simple formatters are great for basic string substitution tags. But if you need to handle tag options, or have access to the parser context or parent tag, you can write a formatter function that returns whatever HTML you like:

```
# A custom render function that uses the tag name as a color style.
def render_color(tag_name, value, options, parent, context):
    return '<span style="color:%s;">%s</span>' % (tag_name, value)
# Installing advanced formatters.
for color in ('red', 'blue', 'green', 'yellow', 'black', 'white'):
    parser.add_formatter(color, render_color)
```

Advanced Quote Example

Suppose you want to support an author option on your quote tags. Your formatting function might look something like this:

```
def render_quote(tag_name, value, options, parent, context):
    author = u''
    # [quote author=Somebody]
    if 'author' in options:
        author = options['author']
    # [quote=Somebody]
    elif 'quote' in options:
        author = options['quote']
    # [quote Somebody]
    elif len(options) == 1:
        key, val = options.items()[0]
        if val:
            author = val
        elif key:
            author = key
    # [quote Firstname Lastname]
    elif options:
        author = ' '.join([k for k in options.keys()])
    extra = '<small>%s</small>' % author if author else ''
    return '<blockquote><p>%s</p>%s</blockquote>' % (value, extra)

# Now register our new quote tag, telling it to strip off whitespace, and the newline␣
↪after the [/quote].
parser.add_formatter('quote', render_quote, strip=True, swallow_trailing_newline=True)
```

Custom Tag Options

When registering a formatter (simple or advanced), you may pass several keyword options for controlling the parsing/rendering behavior.

newline_closes [= False] True if a newline should automatically close this tag.

same_tag_closes [= False] True if another start of the same tag should automatically close this tag.

standalone [= False] True if this tag does not have a closing tag.

render_embedded [= True] True if tags should be rendered inside this tag.

transform_newlines [= True] True if newlines should be converted to markup.

escape_html [= True] True if HTML characters (<, >, and &) should be escaped inside this tag.

replace_links [= True] True if URLs should be replaced with link markup inside this tag.

replace_cosmetic [= True] True if cosmetic replacements (elipses, dashes, etc.) should be performed inside this tag.

strip [= False] True if leading and trailing whitespace should be stripped inside this tag.

swallow_trailing_newline [= False] True if this tag should swallow the first trailing newline (i.e. for block elements).

CHAPTER 4

Indices and tables

- genindex
- modindex
- search