# django-sekizai Documentation

***Release 0.6.1***

**Jonas Obrist**

September 23, 2016

Contents

**Note:** If you get an error when using django-sekizai that starts with **Invalid block tag:**, please read *Restrictions*.

CHAPTER 1

# About

Sekizai means “blocks” in Japanese, and that’s what this app provides. A fresh look at blocks. With django-sekizai you can define placeholders where your blocks get rendered and at different places in your templates append to those blocks. This is especially useful for css and javascript. Your sub-templates can now define css and Javascript files to be included, and the css will be nicely put at the top and the Javascript to the bottom, just like you should. Also sekizai will ignore any duplicate content in a single block.

CHAPTER 2

# Dependencies

- Python 2.7, 3.3, 3.4 or 3.5.
- Django 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9 or 1.10.
- django-classy-tags 0.3.1 or higher.

CHAPTER 3

# Usage

## 3.1 Configuration

In order to get started with django-sekizai, you'll need to do the following steps:

- Put 'sekizai' into your `INSTALLED_APPS` setting.
- **Use one of the following:**
  - For Django versions before 1.10, add `sekizai.context_processors.sekizai` to your `TEMPLATE_CONTEXT_PROCESSORS` setting and use `django.template.RequestContext` when rendering your templates.

    For Django versions after 1.10, add `sekizai.context_processors.sekizai` to your `TEMPLATES['OPTIONS']['context_processors']` setting and use `django.template.RequestContext` when rendering your templates.

  or

  - Use `sekizai.context.SekizaiContext` when rendering your templates.

## 3.2 Template Tag Reference

**Note:** All sekizai template tags require the `sekizai_tags` template tag library to be loaded.

### 3.2.1 Handling code snippets

New in version 0.7: The `strip` flag was added.

Sekizai uses `render_block` and `addtoblock` to handle unique code snippets. Define your blocks using `{% render_block <name> %}` and add data to that block using `{% addtoblock <name> [strip] %}...{% endaddtoblock %}`. If the strip flag is set, leading and trailing whitespace will be removed.

Example Template:

```
{% load sekizai_tags %}

<html>
<head>
```

```
{% render_block "css" %}
</head>
<body>
Your content comes here.
Maybe you want to throw in some css:
{% addtoblock "css" %}
<link href="/media/css/stylesheet.css" media="screen" rel="stylesheet" type="text/css" />
{% endaddtoblock %}
Some more content here.
{% addtoblock "js" %}
<script type="text/javascript">
alert("Hello django-sekizai");
</script>
{% endaddtoblock %}
And even more content.
{% render_block "js" %}
</body>
</html>
```

Above example would roughly render like this:

```
<html>
<head>
<link href="/media/css/stylesheet.css" media="screen" rel="stylesheet" type="text/css" />
</head>
<body>
Your content comes here.
Maybe you want to throw in some css:
Some more content here.
And even more content.
<script type="text/javascript">
alert("Hello django-sekizai");
</script>
</body>
</html>
```

**Note:** It's recommended to have all `render_block` tags in your base template, the one that gets extended by all your other templates.

### 3.2.2 Restrictions

**Warning:** `{% render_block %}` tags **must not** be placed inside a template tag block (a template tag which has an end tag, such as `{% block %}...{% endblock %}` or `{% if %}...{% endif %}`).

**Warning:** `{% render_block %}` tags **must not** be in an included template!

**Warning:** If the `{% addtoblock %}` tag is used in an **extending** template, the tags **must** be placed within `{% block %}...{% endblock %}` tags.

**Warning:** `{% addtoblock %}` tags **must not** be used in a template included with `only` option!

### 3.2.3 Handling data

Sometimes you might not want to use code snippets but rather just add a value to a list. For this purpose there are the `{% with_data <name> as <varname> %}...{% end_with_data %}` and `{% add_data <name> <value> %}` template tags.

Example:

```
{% load sekizai_tags %}

<html>
<head>
{% with_data "css-data" as stylesheets %}
{% for stylesheet in stylesheets %}
    <link href="{{ MEDIA_URL }}{{ stylesheet }}" media="screen" rel="stylesheet" type="text/css" />
{% endfor %}
{% end_with_data %}
</head>
<body>
Your content comes here.
Maybe you want to throw in some css:
{% add_data "css-data" "css/stylesheet.css" %}
Some more content here.
</body>
</html>
```

Above example would roughly render like this:

```
<html>
<head>
<link href="/media/css/stylesheet.css" media="screen" rel="stylesheet" type="text/css" />
</head>
<body>
Your content comes here.
Maybe you want to throw in some css:
Some more content here.
And even more content.
</body>
</html>
```

**Warning:** The restrictions for `{% render_block %}` also apply to `{% with_data %}`, see above. The restrictions for `{% addtoblock %}` also apply to `{% add_data %}`, see above.

### 3.2.4 Sekizai data is unique

All data in sekizai is enforced to be unique within its block namespace. This is because the main purpose of sekizai is to handle javascript and css dependencies in templates.

A simple example using `addtoblock` and `render_block` would be:

```
{% load sekizai_tags %}

{% addtoblock "js" %}
    <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/mootools/1.3.0/mootools
{% endaddtoblock %}

{% addtoblock "js" %}
```

```
    <script type="text/javascript">
        $('firstelement').set('class', 'active');
    </script>
{% endaddtoblock %}

{% addtoblock "js" %}
    <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/mootools/1.3.0/mootools
{% endaddtoblock %}

{% addtoblock "js" %}
    <script type="text/javascript">
        $('secondelement').set('class', 'active');
    </script>
{% endaddtoblock %}

{% render_block "js" %}
```

Above template would roughly render to:

```
<script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/mootools/1.3.0/mootools-yu
<script type="text/javascript">
    $('firstelement').set('class', 'active');
</script>
<script type="text/javascript">
    $('secondelement').set('class', 'active');
</script>
```

New in version 0.5.

### 3.2.5 Processing sekizai data

Because of the restrictions of the `{% render_block %}` tag, it is not possible to use sekizai with libraries such as django-compressor directly. For that reason, sekizai added postprocessing capabilities to `render_block` in version 0.5.

Postprocessors are callable Python objects (usually functions) that get the render context, the data in a sekizai namespace and the name of the namespace passed as arguments and should return a string.

An example for a processor that uses the Django builtin spaceless functionality would be:

```
def spaceless_post_processor(context, data, namespace):
    from django.utils.html import strip_spaces_between_tags
    return strip_spaces_between_tags(data)
```

To use this post processor you have to tell `render_block` where it's located. If above code sample lives in the Python module `myapp.sekizai_processors` you could use it like this:

```
...
{% render_block "js" postprocessor "myapp.sekizai_processors.spaceless_post_processor" %}
...
```

It's also possible to pre-process data in `{% addtoblock %}` like this:

```
{% addtoblock "css" preprocessor "myapp.sekizai_processors.processor" %}
```

CHAPTER 4

# Helpers

## 4.1 sekizai.helpers

**get_namespaces**(*template*)

Returns a list of all sekizai namespaces found in `template`, which should be the name of a template. This method also checks extended templates.

**validate_template**(*template*, *namespaces*)

Returns `True` if all namespaces given are found in the template given. Useful to check that the namespaces required by your application are available, so you can failfast if they're not.

CHAPTER 5

# Example

A full example on how to use django-sekizai and when.

Let's assume you have a website, where all templates extend base.html, which just contains your basic HTML structure. Now you also have a small template which gets included on some pages. This template needs to load a javascript library and execute some specific javascript code.

Your `base.html` might look like this:

```
{% load sekizai_tags %}<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" dir="ltr">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="x-ua-compatible" content="ie=8" />
    <title>Your website</title>
    <link rel="shortcut icon" type="image/x-icon" href="/favicon.ico" />
    <link rel="stylesheet" type="text/css" href="{{ MEDIA_URL }}css/base.css" media="all" />
    <link rel="stylesheet" type="text/css" href="{{ MEDIA_URL }}css/print.css" media="print" />
    {% render_block "css" %}
</head>
<body>
{% block "content" %}
{% endblock %}
<script type="text/javascript" src="{{ MEDIA_URL }}js/libs/jquery-1.4.2.js"></script>
{% render_block "js" %}
</body>
</html>
```

As you can see, we load `sekizai_tags` at the very beginning. We have two sekizai namespaces: "css" and "js". The "css" namespace is rendered in the head right after the base css files, the "js" namespace is rendered at the very bottom of the body, right after we load jQuery.

Now to our included template. We assume there's a context variable called `userid` which will be used with the javascript code.

Your template (`inc.html`) might look like this:

```
{% load sekizai_tags %}
<div class="my-div">
    <ul id="dynamic-content-{{ userid }}"></ul>
</div>

{% addtoblock "js" %}
<script type="text/javascript" src="{{ MEDIA_URL }}js/libs/mylib.js"></script>
{% endaddtoblock %}
```

```
{% addtoblock "js" %}
<script type="text/javascript">
$(document).ready(function(){
    $('#dynamic-content-{{ userid }}').do_something();
}
</script>
{% endaddtoblock %}
```

The important thing to notice here is that we split the javascript into two `addtoblock` blocks. Like this, the library 'mylib.js' is only included once, and the userid specific code will be included once per userid.

Now to put it all together let's assume we render a third template with `[1, 2, 3]` as `my_userids` variable.

The third template looks like this:

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

{% block "content" %}
{% for userid in my_userids %}
    {% include "inc.html" %}
{% endfor %}
{% endblock %}
```

And here's the rendered template:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-stri
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" dir="ltr">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="x-ua-compatible" content="ie=8" />
    <title>Your website</title>
    <link rel="shortcut icon" type="image/x-icon" href="/favicon.ico" />
    <link rel="stylesheet" type="text/css" href="/media/css/base.css" media="all" />
    <link rel="stylesheet" type="text/css" href="/media/css/print.css" media="print" />
</head>
<body>
<div class="my-div">
    <ul id="dynamic-content-1"></ul>
</div>
<div class="my-div">
    <ul id="dynamic-content-2"></ul>
</div>
<div class="my-div">
    <ul id="dynamic-content-3"></ul>
</div>
<script type="text/javascript" src="/media/js/libs/jquery-1.4.2.js"></script>
<script type="text/javascript" src="{{ MEDIA_URL }}js/libs/mylib.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $('#dynamic-content-1').do_something();
}
</script>
<script type="text/javascript">
$(document).ready(function(){
    $('#dynamic-content-2').do_something();
}
</script>
<script type="text/javascript">
$(document).ready(function(){
```

```
    $('#dynamic-content-3').do_something();
}
</script>
</body>
</html>
```

CHAPTER 6

# Changelog

## 6.1 0.10.0

- Added support for Django 1.10
- Removed support for Python 2.6

## 6.2 0.9.0

- Added Changelog
- Added support for Django 1.9
- Added support for Python 3.5

## G



## V