

---

# **django-jstemplate Documentation**

*Release 1.3.8*

**Mjumbe Wawatu Ukweli**

March 01, 2017



<b>1</b>	<b>Quickstart</b>	<b>3</b>
1.1	Dependencies . . . . .	3
1.2	Installation . . . . .	3
1.3	Usage . . . . .	3
1.4	An Example . . . . .	4
1.5	Handlebars.js . . . . .	4
1.6	What's going on? . . . . .	5
1.7	Flavors of Javascript templates . . . . .	5
1.8	Matching Multiple Template Files . . . . .	6
<b>2</b>	<b>Internationalization (i18n)</b>	<b>7</b>
2.1	Under the hood . . . . .	7
<b>3</b>	<b>Settings</b>	<b>9</b>
<b>4</b>	<b>Advanced usage</b>	<b>11</b>
4.1	Custom Finders . . . . .	11
4.2	Custom Preprocessors . . . . .	11
4.3	Custom Flavors . . . . .	12
<b>5</b>	<b>Source</b>	<b>13</b>
<b>6</b>	<b>Running tests</b>	<b>15</b>



A templatetag framework for easier integration of [mustache.js](#), [dust.js](#), [handlebars.js](#), or other JavaScript templates with Django templates. Also will wrap your templates in elements expected for libraries such as [ICanHaz.js](#). Django-jstemplate is extensible, so if your favorite template library is not included, it's easy to add. Inspired by [django-icanhaz](#).



---

## Quickstart

---

### Dependencies

Tested with Django 1.5 through trunk, and Python 2.6, 2.7, and 3.3. Almost certainly works with older versions of both.

### Installation

Install from PyPI with pip:

```
pip install django-jstemplate
```

or get the in-development version:

```
pip install django-jstemplate==dev
```

### Usage

- Add "jstemplate" to your INSTALLED\_APPS setting.
- In your HTML header, include the Javascript templating library of your choice (I like to use mustache.js):

```
wget https://raw.githubusercontent.com/janl/mustache.js/master/mustache.js
mv mustache.js app/static/libs/
```

- `{% load jstemplate %}` and use `{% mustachejs "templatename" %}` in your Django templates to safely embed the mustache.js template at `<JSTEMPLATE_DIRS-entry>/templatename.html` into your Django template. It will be stored in the `Mustache.TEMPLATES` object as a string, accessible as `Mustache.TEMPLATES.templatename`.
- In your JavaScript, use `Mustache.to_html(Mustache.TEMPLATES.templatename, {...}, Mustache.TEMPLATES)` to render your mustache template. Alternatively, if you include the `libs/django.mustache.js` script in your HTML, you can use `Mustache.template('templatename').render({...})` to render your mustache template.

## An Example

For example consider the files `app/jstemplates/main.mustache`:

```
<div>
  <p>This is {{ name }}'s template</p>
</div>
```

and `app/templates/main.html`:

```
{% load jstemplate %}

<html>
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.js"></script>

  <script src="{{ STATIC_URL }}libs/mustache.js"></script>
  <script src="{{ STATIC_URL }}libs/django.mustache.js"></script>
</head>

<body>
  <div id="dynamic-area"></div>

  {% mustachejs "main" %}

  <script>
    $(document).ready(function() {

      var $area = $('#dynamic-area')
        , template;

      // Either render by accessing the TEMPLATES object
      // directly...

      $area.html(Mustache.to_html(Mustache.TEMPLATES.main));

      // ...or render by using a cached template object
      // (requires django.mustache.js)

      template = Mustache.template('main');
      $area.html(template.render());

    });
  </script>
</body>
</html>
```

## Handlebars.js

The `handlebarsjs` template tag has two optional parameters.

- *precompile* will add your templates to a `templates` attribute on the `Handlebars` object. For example:

```
{% handlebarsjs 'my-template' precompile %}

<script>
```



```
var html = Handlebars.templates['my-template']();
</script>
```

- `register_partial` will make your templates available as partials to use in other templates. For example, take the following templates:

*my-template.html:*

```
<p>This is a list</p>
<ul>
  {{# each people }}
  {{> my-partial }}
  {{/ each }}
</ul>
```

*my-partial.html:*

```
<li>{{ name }}</li>
```

*index.html:*

```
...
{% handlebarsjs 'my-template' %}
{% handlebarsjs 'my-partial' register_partials %}
...
```

## What's going on?

Any time you use the `mustachejs` template tag, or any of the other `jstemplate` tags:

```
{% load jstemplate %}
{% mustachejs "main" %}
```

django-jstemplate will generate something like the following:

```
<script>Mustache.TEMPLATES=Mustache.TEMPLATES||{};Mustache.TEMPLATES['main']='<div>\n <p>This is {{
```

This stores the text of the template in an attribute on the `Mustache.TEMPLATES` object (it will first create the object if it does not yet exist). The `Mustache.template(...)` function then creates an object with a `render(...)` method that has a similar signature as `Mustache.to_html(...)`, except without the template name as the first parameter. The `render` method will also use the set of templates in `Mustache.TEMPLATES` as partials, allowing any template that django-mustachejs knows about to be used as a template partial as well.

Other tags, just as `{% icanhazjs %}` wrap your template in the elements expected for particular libraries (like for [ICanHaz.js](#) in that situation).

## Flavors of Javascript templates

In addition to `{% mustachejs ... %}`, django-jstemplate comes with several template tags that you can use to render your mustache templates:

- `{% dustjs ... %}` renders templates ready for consumption by `dust.js`
- `{% icanhazjs ... %}` renders templates ready for consumption by `ICanHaz.js`
- `{% handlebarsjs ... %}` renders templates ready for consumption by `Handlebars.js`

- `{% rawjstemplate ... %}` renders the raw contents of a mustache template, after preprocessing

## Matching Multiple Template Files

The name provided to the template tag can be a string that will match a single file, a file glob pattern, or a regular expression. Using the template tag `{% mustachejs [glob/regex] %}` in your Django templates will embed all files matching that regex in the template directories. So, `{% mustachejs '(.*)_template' %}` and `{% mustachejs '*_template' %}` would both match *note\_template.html* and *comment\_template.html*, giving them templatenames *note\_template* and *comment\_template*, respectively. (Note that the regular expression pattern must contain parentheses denoting a single matching group; this group will become the name of the template).

---

## Internationalization (i18n)

---

django-mustachejs supports internationalization tags. In your settings module, set the `JSTEMPLATE_I18N_TAGS` variables (default: `('_', 'i18n')`). These tags can be used to preprocess the javascript templates into translatable content. For example:

```
<div>{{#_}}Hello, {{name}}. I like your {{color}} {{thing}}?{{/_}}</div>
```

may render to:

```
<div>Salut, {{name}}. J'aime votre {{thing}} {{color}}?</div>
```

The translatable strings will be picked up by Django's `makemessages` management command.

### Under the hood

In order to avoid having to send our project's translation mapping to the client, we have built-in the ability to preprocess `i18n` tags in the mustache templates.

There aren't any nice solutions here. The code behind `makemessages` unfortunately isn't extensible, so we can:

- Duplicate the command + code behind it.
- Offer a separate command for Mustache tag extraction.
- Try to get Django to offer hooks into `makemessages`.
- Monkey-patch.

We are currently doing that last thing. In this case we override the `templatize` method. `templatize` takes a template, extracts the translatable strings (along with desired metadata), and generates a file that `xgettext` knows how to parse, e.g. a file with Python syntax. We override this function to find Mustache-tagged strings if the file that we are templating is in one of the paths found by the active `JSTEMPLATE_FINDERS`.



---

## Settings

---

- Set `JSTEMPLATE_FINDERS` to configure the dotted class names of the finders the application will use. By default, this is the following list:

```
["jstemplate.finders.FilesystemFinder",  
 "jstemplate.finders.AppFinder",  
 "jstemplate.finders.FilesystemRegexFinder",  
 "jstemplate.finders.AppRegexFinder",]
```

- Set the `JSTEMPLATE_DIRS` setting to a list of full (absolute) path to directories where you will store your mustache templates. By default this is an empty list.
- Set `JSTEMPLATE_APP_DIRNAMES` to a list of directory names that can be found under directories of applications specified in `INSTALLED_APPS`. By default, this setting has the value of `["jstemplates"]`.
- Set the `JSTEMPLATE_EXTS` setting to a list of the app should search for to find template files. By default this is set to `['mustache', 'html']`. Order matters (e.g., `*.mustache` will take precedence over `*.html`).
- Set the `JSTEMPLATE_PREPROCESSORS` variable to control how the templates are preprocessed. By default, there is one preprocessor activated:

```
['jstemplate.preprocessors.I18nPreprocessor']
```

The `I18nPreprocessor` will translate marked strings before rendering the template. To disable this feature, set `JSTEMPLATE_PREPROCESSORS` to an empty list.

- Set `JSTEMPLATE_I18N_TAGS` to the names of the tags used to mark strings for internationalization. By default, this is set to the list:

```
["_", "i18n"]
```

Meaning that text falling between the tags `{{#_}}...{{/_}}` and `{{#i18n}}...{{/i18n}}` will be marked for translation.



---

## Advanced usage

---

### Custom Finders

The finding of templates can be fully controlled via the `JSTEMPLATE_FINDERS` setting, which is a list of dotted paths to finder classes. A finder class should be instantiable with no arguments, and have a `find(name)` method which returns either (1) the full absolute path to a template file, given a base-name, or (2) a list of (template name, template file path) pairs according to the given base name.

By default, `JSTEMPLATE_FINDERS` contains `"jstemplate.finders.FilesystemFinder"` (which searches directories listed in `JSTEMPLATE_DIRS`), `"jstemplate.finders.AppFinder"` (which searches subdirectories named in `JSTEMPLATE_APP_DIRNAMES` of each app in `INSTALLED_APPS`), `"jstemplate.finders.FilesystemRegexFinder"`, and `"jstemplate.finders.AppRegexFinder"`, in that order – thus templates found in `JSTEMPLATE_DIRS` take precedence over templates in apps, and templates identified by file glob patterns take precedence over those identified by regular expression patterns.

### Custom Preprocessors

Before your JavaScript templates are placed into your Django templates, they are run through preprocessors. By default, the only preprocessor enabled is for *internationalization (i18n)*. The `i18n` preprocessor finds all text between `{{#_}}` and `{{/_}}`, translates it with `gettext`, and inserts the translated text into the template, stripping the `{{#_}}` and `{{/_}}` tags.

You can build your own preprocessors as well. A good use would be to do things like including generated URLs in your templates. For example, in your template, when you have `{{reverse_url 'my_url_name'}}`, you might want to run that through Django's `reverse` method.

A preprocessor class is pretty simple. All it requires is a method with the following signature:

```
def process(self, content):  
    ...
```

Where `content` is the actual text of the JS template. Then, just add the dotted name of your class to the `JSTEMPLATE_PREPROCESSORS` settings variable.

## Custom Flavors

It is simple to extend `django-jstemplate` to prepare your JavaScript templates to be used with your favorite Javascript library by creating a template node class that derives from `jstemplate.templatetags.BaseJSTemplateNode`, and overriding a single function. Refer to the existing tag definitions for `mustachejs`, `icanhazjs`, `rawjstemplate`, and `handlebarsjs` for more information.



---

**Source**

---

The source for `django-jstemplate` is available on [GitHub](#)



---

## Running tests

---

To run the tests (for development), install `mock` and `six` and run:

```
jstemplate/tests/project/manage.py test
```