
django-missing Documentation

Release 0.1.11

wlan slovenija

Sep 27, 2017

Contents

1	Contents	3
1.1	Middleware	3
1.2	Template Tags and Filters	3
1.3	Views	6
1.4	Debugging	6
1.5	Timezone	6
1.6	JavaScript	7
2	Source Code and Issue Tracker	9
3	Indices and Tables	11
	Python Module Index	13

This Django application bundles some common and useful features which have not (yet) found a way into Django itself.

You are invited to contribute your own features so that together we create a powerful and good library of Django additions. Because sometimes you cannot imagine an useful feature before somebody shows it to you. So it is useful to have all of them collected together.

Middleware

class `missing.middleware.ForceAdminLanguage`

Middleware which forces language in Django admin to `ADMIN_LANGUAGE_CODE` setting value.

Useful when not wanting that Django content language interferes with admin language, especially when admin interface is not translated fully in all languages content is, or when error messages in admin interface are hard to debug because of a rare language they are displayed in.

Should be added to `MIDDLEWARE_CLASSES` after `LocaleMiddleware` middleware:

```
MIDDLEWARE_CLASSES = (  
    ...  
    django.middleware.locale.LocaleMiddleware,  
    missing.middleware.ForceAdminLanguage,  
    ...  
)
```

Template Tags and Filters

`context_tags` module

Use `{% load context_tags %}` in your template to load this module.

`missing.template_tags.context_tags.setcontext` (*parser, token*)

Sets (updates) current template context with the rendered output of the block inside tags.

This is useful when some template tag does not support storing its output in the context itself or we need some complex content (like language, user or URL dependent content) multiple times.

Example usage:

```
{% setcontext as varname %}
    {% complextag %}
{% endsetcontext %}
```

lang_tags module

Use `{% load lang_tags %}` in your template to load this module.

`missing.templatetags.lang_tags.translate(string, lang_code)`

Translates given string to the specified language.

This is useful for text you need in some other language than the current language. For example, for links inviting users to switch to their language.

Example usage:

```
{% translate "Do you understand this?" "de" %}
```

list_tags module

Use `{% load list_tags %}` in your template to load this module.

`missing.templatetags.list_tags.divide_list(value, count)`

Divides input list into the given number of sublists.

Last sublist can be shorter if input list length is not a multiplier of the given number of sublists.

Example usage:

```
<tr>
{% for column in objects|divide_list:"2" %}
    <td><ul>
        {% for obj in column %}
            <li>{{ obj }}</li>
        {% endfor %}
    </ul></td>
{% endfor %}
</tr>
```

`missing.templatetags.list_tags.split_list(value, length)`

Splits input list into sublists of the given length.

Last sublist can be shorter if input list length is not a multiplier of the given length.

Example usage:

```
{% for group in objects|split_list:"4" %}
    <tr>
        {% for obj in group %}
            <td>{{ obj }}</td>
        {% endfor %}
    </tr>
{% endfor %}
```


string_tags module

Use `{% load string_tags %}` in your template to load this module.

`missing.templatetags.string_tags.count` (*value*, *arg*)

Returns the number of non-overlapping occurrences of an argument substring in the given string.

`missing.templatetags.string_tags.ensure_sentence` (**args*, ***kwargs*)

Ensures that string ends with dot if it does not already end with some punctuation.

`missing.templatetags.string_tags.startswith` (*value*, *arg*)

Returns True if the given string starts with an argument prefix, otherwise returns False.

url_tags module

Use `{% load url_tags %}` in your template to load this module.

`missing.templatetags.url_tags.fullurl` (*parser*, *token*)

Builds an absolute (full) URL from the given location and the variables available in the request.

If no location is specified, the absolute (full) URL is built on `django.http.HttpRequest.get_full_path()`.

It is a wrapper around `django.http.HttpRequest.build_absolute_uri()`. It requires request to be available in the template context (for example, by using `django.core.context_processors.request` context processor).

Example usage:

```
{% url "view_name" as the_url %}
{% fullurl the_url %}
```

`missing.templatetags.url_tags.slugify2` (**args*, ***kwargs*)

Normalizes string, converts to lowercase, removes non-alpha characters, and converts spaces to hyphens.

It is similar to built-in `slugify` but it also handles special characters in variety of languages so that they are not simply removed but properly transliterated/downcoded.

`missing.templatetags.url_tags.urltemplate` (*viewname*, **args*, ***kwargs*)

Creates URI template in a similar way to how url tags work but leaving parts of a URI to be filled in by a client. See [RFC 6570](#) for more information.

Names of parts are taken from named groups in URL regex pattern used for the view, or as a part's sequence number (zero-based) for unnamed groups. You can pre-fill some parts by specifying them as additional arguments to the tag.

Warning: Tag cannot check if pre-fill values specified will really match back the URL regex pattern, so make sure yourself that they do.

Example usage:

```
{% with variable="42" %}
  {% urltemplate "view_name" arg1="value" arg2=variable %}
{% endwith %}
```

If URL pattern would be defined like:

```
url(r'^some/view/(?P<arg1>.*)/(?P<arg2>.*)/(?P<param>.*)/$', some_view, name=
↳ 'view_name'),
```

The result would be:

```
/some/view/value/42/{param}/
```

Note: Requires Django 1.4+.

Views

class `missing.views.EnsureCsrfCookieMixin`

Mixin for Django class-based views which forces a view to send the CSRF cookie.

This is useful when using Ajax-based sites which do not have an HTML form with a `csrf_token` that would cause the required CSRF cookie to be sent.

`missing.views.forbidden_view(request, reason='')`

Displays 403 forbidden page. For example, when request fails CSRF protection.

Similar to Django built-in view, but using template and request context.

Debugging

class `missing.debug.SafeExceptionReporterFilter`

Safe exception reporter filter which also filters password from request environment (META).

This is useful to not display password and other sensitive data passed to Django through its process environment.

Furthermore, it configures Django to additionally clean settings with URL, CSRF, and COOKIE in keys.

To install it, configure Django to:

```
DEFAULT_EXCEPTION_REPORTER_FILTER = 'missing.debug.SafeExceptionReporterFilter'
```

and import `missing.debug` somewhere in your code, for example, in `urls.py` of your project.

Note: Requires Django 1.4+.

Timezone

`missing.timezone.to_date(value)`

Function which knows how to convert timezone-aware `datetime` objects to `date` objects, according to guidelines from [Django documentation](#).

JavaScript

Automatic Slug Generation

Django admin and other Django applications can have slug fields which are automatically updated/generated in user's browser using JavaScript. Django bundles JavaScript code necessary for this but it behaves differently than built-in `slugify` template filter in Python. For this reason `django-missing` provides JavaScript code with equal functionality, implemented directly in JavaScript. It comes in two flavors:

`slugify` equivalent

A JavaScript equivalent to built-in `slugify` template filter. You can load by adding something like this in your page (or Django admin) `<head>` section (in template):

```
<script type="text/javascript" src="{ STATIC_URL }missing/nlldata.js"></script>
<script type="text/javascript" src="{ STATIC_URL }missing/nl1n.js"></script>
<script type="text/javascript" src="{ STATIC_URL }missing/urlify.js"></script>
```

Of course above mentioned files should be published by your Django site installation.

`slugify2` equivalent

If you want to use improved `slugify2()` template filter in Python, you can also use its equivalent in JavaScript:

```
<script type="text/javascript" src="{ STATIC_URL }missing/nlldata.js"></script>
<script type="text/javascript" src="{ STATIC_URL }missing/nl1n.js"></script>
<script type="text/javascript" src="{ STATIC_URL }missing/urlify2.js"></script>
```

Datetime Formatting

Once you load [internationalization in JavaScript code](#), Django provides `get_format` function to access configured [datetime and other formats](#) but it is lacking function to format JavaScript Date objects according to those formats. By loading:

```
<script type="text/javascript" src="{ STATIC_URL }missing/date.js"></script>
```

JavaScript Date prototype is extended with `strfdate` method:

```
new Date().strfdate(get_format('DATETIME_FORMAT'))
```

Note, to format datetime input formats (those using `%` for placeholders) Django admin provides limited support through its `strftime` method added to JavaScript Date prototype when loading:

```
<script type="text/javascript" src="{ STATIC_URL }admin/js/core.js"></script>
```

and use, for example, as:

```
new Date().strftime(get_format('DATE_INPUT_FORMATS')[0])
```

Relative datetime

JavaScript implementations of `timesince`, `timeuntil`, and `naturaltime` Django filters are also available as extensions to JavaScript `Date` prototype by loading:

```
<script type="text/javascript" src="{% STATIC_URL %}missing/humanize.js"></script>
```

Additionally, `updatingNaturaltime` method is provided which behaves similarly to `naturaltime` method but it takes a DOM element or jQuery selector as an optional argument and makes sure it is updated as time progresses.

CHAPTER 2

Source Code and Issue Tracker

[GitHub](#) is used for development, so source code and issue tracker is found there. Feel free to fork and contribute with pull requests, or open issues you might have. Ideas for improvements are welcome, too.

CHAPTER 3

Indices and Tables

- `genindex`
- `modindex`
- `search`

m

- `missing.debug`, 6
- `missing.middleware`, 3
- `missing.templatetags.context_tags`, 3
- `missing.templatetags.lang_tags`, 4
- `missing.templatetags.list_tags`, 4
- `missing.templatetags.string_tags`, 5
- `missing.templatetags.url_tags`, 5
- `missing.timezone`, 6
- `missing.views`, 6

C

count() (in module missing.templatetags.string_tags), 5

D

divide_list() (in module missing.templatetags.list_tags), 4

E

ensure_sentence() (in module missing.templatetags.string_tags), 5

EnsureCsrfCookieMixin (class in missing.views), 6

F

forbidden_view() (in module missing.views), 6

ForceAdminLanguage (class in missing.middleware), 3

fullurl() (in module missing.templatetags.url_tags), 5

M

missing.debug (module), 6

missing.middleware (module), 3

missing.templatetags.context_tags (module), 3

missing.templatetags.lang_tags (module), 4

missing.templatetags.list_tags (module), 4

missing.templatetags.string_tags (module), 5

missing.templatetags.url_tags (module), 5

missing.timezone (module), 6

missing.views (module), 6

R

RFC

RFC 6570, 5

S

SafeExceptionReporterFilter (class in missing.debug), 6

setcontext() (in module missing.templatetags.context_tags), 3

slugify2() (in module missing.templatetags.url_tags), 5

split_list() (in module missing.templatetags.list_tags), 4

startswith() (in module missing.templatetags.string_tags), 5

T

to_date() (in module missing.timezone), 6

translate() (in module missing.templatetags.lang_tags), 4

U

urltemplate() (in module missing.templatetags.url_tags), 5