## Contents

1 Tutorial 3
   1.1 Getting started .................................................. 3
   1.2 Basic example .................................................... 3
   1.3 *forme* tags hierarchy ....................................... 4
   1.4 Use or replace .................................................. 4

2 Tag reference 7
   2.1 Usage of paired tags ......................................... 7
   2.2 Tag hierarchy .................................................. 7
   2.3 General tag syntax ............................................ 9

3 Performance 11

4 Just informative 13

5 What’s this 15

6 Working draft 17

7 Indices and tables 19
| Warning: This is documentation to first pre release. Most of described functionality isn’t implemented yet. |

Contents:
The aim of `django-forme` is to allow customization of form rendering in templates rather than in `forms.py` (or other Python source file). Define default templates using your css framework markup and only alter rendering of those fields which need special markup.

### 1.1 Getting started

1. Install `django-forme` using `pip`:

   ```bash
   pip install django-forme
   ```

2. Add `forme` to `INSTALLED_APPS`:

   ```python
   INSTALLED_APPS = (  
     ...  
     'forme',  
   )
   ```

3. Load `forme` tags in template:

   ```html
   {% load forme %}
   ```

### 1.2 Basic example

Suppose we have following login form:

```python
class LoginForm(forms.Form):
    username = forms.CharField()
    password = forms.CharField(widget=forms.PasswordInput)
```

The most trivial use case is rendering whole form using default template. Instead of writing `{{ form }}` in template, we use `forme` tag:

```html
{% forme form %}
```

Result would be:

```html
<label for="id_username">Username</label>
<input type="text" name="username" id="id_username">
<label for="id_password">Password</label>
<input type="password" name="password" id="id_password">
```
We can specify default template to render our form using \texttt{template} keyword:

\begin{verbatim}
{% forme form template="bootstrap" %}
\end{verbatim}

\begin{verbatim}
<div class="form-group">
    <label for="id_username">Username</label>
    <input type="text" class="form-control" id="id_username" name="username">
</div>
<div class="form-group">
    <label for="id_password">Password</label>
    <input type="password" class="form-control" id="id_password" name="password">
</div>
\end{verbatim}

... and that is basically all we can do with \texttt{forme} tag only. More real use cases come when we introduce other template tags.

### 1.3 \texttt{forme} tags hierarchy

Forms in general are structured in hierarchy:

\begin{verbatim}
<form>
    <hidden fields>
    <non-field errors>
    <fieldset>
        <field>
            <label>
                <input>
            </label>
        </field>
        <field errors>
    </fieldset>
</form>
\end{verbatim}

Field errors and non-field errors are relevant for bound forms only. Grouping fields into fieldsets is optional. This structure represents also \texttt{forme} tags hierarchy with corresponding tags:

<table>
<thead>
<tr>
<th>Element</th>
<th>\texttt{forme} tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>form</td>
<td>{% forme %}</td>
</tr>
<tr>
<td>hidden fields</td>
<td>{% hiddenfields %}</td>
</tr>
<tr>
<td>non-field errors</td>
<td>{% nonfielderrors %}</td>
</tr>
<tr>
<td>fieldset</td>
<td>{% fieldset %}</td>
</tr>
<tr>
<td>field</td>
<td>{% field %}</td>
</tr>
<tr>
<td>label</td>
<td>{% label %}</td>
</tr>
<tr>
<td>input</td>
<td>{% input %}</td>
</tr>
<tr>
<td>field errors</td>
<td>{% errors %}</td>
</tr>
</tbody>
</table>

Each tag can be written either as \textit{paired} or \textit{unpaired} one depending on context.

Usage of these tags will be discussed in following sections. This definition was mentioned to clarify meaning and structure of \texttt{forme} tags.

### 1.4 Use or replace

Consider following example:
and the result:

```html
<label for="id_username">Username</label>
<input type="text" name="username" id="id_username" />
<label for="id_password">Password</label>
<input class="password" type="password" name="password" id="id_password" />
```

forme tag contains new keyword `replace`. It says: “Replace parts of default template with following templates”. It also makes `forme` tag paired one ending with `{% endforme %}`. Everything between will be considered as an form element template.

The first (and only) tag inside is `input password`. This tag contains `using` keyword which says: “Use this html code as an template to render input.”

In other words: `forme` tag will render two fields, `username` and `password`. `Username` will render label and input using default templates, but `password` will render label using default template and input using template specified inside `input "password"` tag.
django-forme provides eight tags:

**forme** Top level tag which renders form or forms.

**nonfielderrors** Renders form’s non field errors.

**hiddenfields** Renders hidden fields.

**fieldset** Renders set of fields. By default it’s just wrapper around all rows.

**row** Renders one field row consisting of errors, label and field.

**errors** Renders field errors.

**label** Renders label associated with field.

**field** Renders field itself.

### 2.1 Usage of paired tags

Each of tags above can be used as paired or single tag depending on parameters. In general, paired tags are used to specify templates, while unpaired tags are used to render directly using default templates.

Examples:

- Renders form using default template:

  ```
  {% forme form %}
  ```

- Renders form using custom template:

  ```
  {% forme form using %}
  {% nonfielderrors %}
  {% hiddenfields %}
  <div class="inner-form">
    {% fieldset %}
  </div>
  {% endforme %}
  ```

### 2.2 Tag hierarchy

Default templates are required to define tags in following hierarchy:
It’s not allowed to include parent tags under child ones (e.g. `row` within `field` isn’t allowed). However, it is allowed to define child tags on higher levels than in diagram shown above. These tags are considered as template definitions:

```
{% forme form replace %}
{% fieldset "username email" replace %}
  {% field using %}
    <div class="login_field">
      ...
    </div>
  {% endfield %}
{% endfieldset %}

{% fieldset "password password1" replace %}
  {% field using %}
    <div class="password_field">
      ...
    </div>
  {% endfield %}
{% endforme %}
```

In the example above, fields `username` and `email` will be rendered with template defined by `field` tag under `{% fieldset "username email" replace %}` tag. Fields `password` and `password1` will be rendered using different template defined by `field` tag under second `fieldset` tag. The difference between keywords `using` and `replace` is described below.

Rendered html might look like this:

```
<div class="row">
  <div class="login_field">
    <input name="username" ... />
  </div>
</div>
<div class="row">
  <div class="login_field">
    <input name="email" ... />
  </div>
</div>
<div class="row">
  <div class="password_field">
    <input name="password" ... />
  </div>
</div>
```
2.3 General tag syntax

Each of eight tags accepts the same parameters. General valid format is:

```html
{% tag [target [target2] [.....]] [using|replace] %}
```

where:

- **tag** is valid tag name as described in *Tag reference*
- **target** is target to render. It’s either form to render (for `forme` tag) or field(s). While form must be variable containing django form object, fields can be specified either by template variables or string. Multiple fields can be specified using space separated string. Tags below are equal:

```html
{% field "username password" %}
{% field "username" "password" %}
```

- **action** The default action is “render tag using default template”. When `replace` or `using` keyword are set, defined templates override the default ones. Difference between these two actions is detaily described below.

**Note**: When action `using` or `replace` is set, the tag is considered as **paired** one.

Tag without any parameters just defines placeholder, where the tag will be rendered using information available in template context. Only exception is `forme` tag, which always requires either target or action.

2.3.1 Difference between using and replace

Both actions are used to override default templates. When `replace` is used, only child tags templates are overriden, otherwise is overriden also template of tag itself.

Consider following default template:

```html
{% forme using %}
{% fieldset using %
    {%fieldset
        {%row %}
            {%errors %}
            {%label %}
            {%fields %}
        {%endrow %}
    {%endfieldset %}
{%row using %}
</div>
{%endrow %}
```

2.3. General tag syntax
When this template is used to render form using bellow template:

```python
{% forme form replace %}
{% fieldset replace %}
{% row using %}
  <div class="custom row">
    {% errors %}
    {% label %}
    {% fields %}
  </div>
{% endrow %}
{% endfieldset %}
{% endforme %}
```

the output for form with one field might look like this:

```html
<fieldset>
  <div class="custom row">
    ...
  </div>
</fieldset>
```

1. `<fieldset>...</fieldset>` tag is preserved, because `{% fieldset replace %}` overrides only child tags templates (only `row` tag in this case) keeping fieldset template untouched.

2. `row` tag template is overriden, because `{% row using %}` overrides also row template.

Also note, the same output we would get even with this template:

```python
{% forme form replace %}
{% row using %}
  <div class="custom row">
    {% errors %}
    {% label %}
    {% fields %}
  </div>
{% endrow %}
{% endforme %}
```
Profiling tests are disabled by default. To run them use:

$ py.test -m profiling -s

Testing of first two working templates revealed that parsing is about 40x slower, but rendering is only 1.7x slower which is acceptable, since most production settings uses template caching.

<table>
<thead>
<tr>
<th>Engine</th>
<th>Parsing</th>
<th>Rendering</th>
</tr>
</thead>
<tbody>
<tr>
<td>django</td>
<td>0.094 ms</td>
<td>0.636 ms</td>
</tr>
<tr>
<td>forme</td>
<td>3.029 ms</td>
<td>1.077 ms</td>
</tr>
</tbody>
</table>
Comparison above is simple guess. Tested django template is basic {{ form }} which has almost zero parsing time. Also rendering isn’t done by template engine but directly in Form class. Sometimes it doesn’t produce the same result either. It’s only informative comparison and check that there’s no severe performance overhead.
**What’s this**

django-forme is set of template tags allowing to customize django form’s output. The key idea is that most of customization should be done in template, not in form definition. It also implements form styles which defines default form output based on used template (eg. bootstrap, foundation, etc.)
Below are examples showing the planned functionality of app.

- Render form using default template:

  ```
  {% forme form %}
  {% endforme %}
  ```

- Render multiple forms (user_form, profile_form):

  ```
  {% forme user_form profile_form %}
  {% endforme %}
  ```

- Replace template for single field:

  ```
  {% forme form replace %}
  {% field "password" using %}
    <input name="{{ field.html_name }}" id="{{ field.html_id }}"
      type="text" value="{{ field.value }}" />
  {% endfield %}
  {% endforme %}
  ```

- Override order of fields:

  ```
  {% forme form replace %}
  {% field "password" %}
  {% field "username" %}
  {% endforme %}
  ```

- Split fields into fieldsets:

  ```
  {% forme form replace %}
  <div class="column-left">
    {% fieldset "username email" %}
    {% endfieldset %}
  </div>
  <div class="column-right">
    {% fieldset "password password1" %}
    {% endfieldset %}
  </div>
  {% endforme %}
  ```
Indices and tables

- genindex
- modindex
- search