Working with form templates

All you need to do to get your form into a template is to place the form instance into the template context.

So, if your form is called form in the context, {{ form }} will render its <label> and <input> elements

appropriately.

## Additional form template furniture

Don’t forget that a form’s output does not include the surrounding **<form>** tags, or the form’s submit control.

You will have to provide these yourself

# Reusable form templates

The HTML output when rendering a form is itself generated via a template. You can control this by creating

an appropriate template file and setting a custom FORM\_RENDERER to use that **form\_template\_name** site-wide.

You can also customize per-form by overriding the form’s **template\_name** attribute to render the form using

the custom template, or by passing the template name directly to **Form.render().**

The example below will result in {{ form }} being rendered as the output of the ***form\_snippet.html*** **template**.

In your templates:

# In your template:

{{ form }}

# In form\_snippet.html:

{% for field in form %}

<div class="fieldWrapper">

{{ field.errors }}

{{ field.label\_tag }} {{ field }}

</div>

{% endfor %}

Then you can configure the **FORM\_RENDERER** setting:Listing 14: settings.py

from django.forms.renderers import TemplatesSetting

class CustomFormRenderer(TemplatesSetting):

form\_template\_name = "form\_snippet.html"

FORM\_RENDERER = "project.settings.CustomFormRenderer"

. . . or for a single form:

class MyForm(forms.Form):

template\_name = "form\_snippet.html"

...

. . . or for a single render of a form instance, passing in the template name to the Form.render(). Here’s an

example of this being used in a view:

def index(request):

form = MyForm()

rendered\_form = form.render("form\_snippet.html")

context = {"form": rendered\_form}

return render(request, "index.html", context)

# Reusable field group templates

Each field is available as an attribute of the form, using {{ form.name\_of\_field }} in a template. A field

has a as\_field\_group() method which renders the related elements of the field as a group, its label, widget,

errors, and help text.

This allows generic templates to be written that arrange fields elements in the required layout. For example:

{{ form.non\_field\_errors }}

<div class="fieldWrapper">

{{ form.subject.as\_field\_group }}

</div>

<div class="fieldWrapper">

{{ form.message.as\_field\_group }}

</div>

<div class="fieldWrapper">

(continues on next page)

{{ form.sender.as\_field\_group }}

</div>

<div class="fieldWrapper">

{{ form.cc\_myself.as\_field\_group }}

</div>

By default, Django uses the "**django/forms/field.html**" template which is designed for use with the default

"django/forms/div.html" form style.

The default template can be customized by setting field\_template\_name in your project-level

**FORM\_RENDERER:**

from django.forms.renderers import TemplatesSetting

class CustomFormRenderer(TemplatesSetting):

field\_template\_name = "field\_snippet.html"

. . . or on a single field:

class MyForm(forms.Form):

subject = forms.CharField(template\_name="my\_custom\_template.html")

...

. . . or on a per-request basis by calling BoundField.render() and supplying a template name:

def index(request):

form = ContactForm()

subject = form["subject"]

context = {"subject": subject.render("my\_custom\_template.html")}

return render(request, "index.html", context)

# Rendering fields manually

More fine-grained control over field rendering is also possible. Likely this will be in a custom field template,

to allow the template to be written once and reused for each field. However, it can also be directly accessed

from the field attribute on the form. For example:

{{ form.non\_field\_errors }}

<div class="fieldWrapper">

{{ form.subject.errors }}

<label for="{{ form.subject.id\_for\_label }}">Email subject:</label>

{{ form.subject }}

</div>

<div class="fieldWrapper">

{{ form.message.errors }}

<label for="{{ form.message.id\_for\_label }}">Your message:</label>

{{ form.message }}

</div>

<div class="fieldWrapper">

{{ form.sender.errors }}

<label for="{{ form.sender.id\_for\_label }}">Your email address:</label>

{{ form.sender }}

</div>

<div class="fieldWrapper">

{{ form.cc\_myself.errors }}

<label for="{{ form.cc\_myself.id\_for\_label }}">CC yourself?</label>

{{ form.cc\_myself }}

</div>

Complete <label> elements can also be generated using the **label\_tag().** For example:

<div class="fieldWrapper">

{{ form.subject.errors }}

{{ form.subject.label\_tag }}

{{ form.subject }}

</div>

# Rendering form error messages - - -

The price of this flexibility is a bit more work. Until now we haven’t had to worry about how to display form

errors, because that’s taken care of for us. In this example we have had to make sure we take care of any

errors for each field and any errors for the form as a whole. Note **{{ form.non\_field\_errors }}** at the top

of the form and the template lookup for errors on each field.

Using **{{ form.name\_of\_field.errors }}** displays a list of form errors, rendered as an unordered list. This

might look like:

<ul class="errorlist">

<li>Sender is required.</li>

</ul>

The list has a CSS class of **errorlist** to allow you to style its appearance. If you wish to further customize

the display of errors you can do so by looping over them:{% if form.subject.errors %}

<ol>

{% for error in form.subject.errors %}

<li><strong>{{ error|escape }}</strong></li>

{% endfor %}

</ol>

{% endif %}

Non-field errors (and/or hidden field errors that are rendered at the top of the form when using helpers like

**form.as\_p())** will be rendered with an additional class of **nonfield** to help distinguish them from **field specific** errors. For example, **{{ form.non\_field\_errors }}** would look like:

<ul class="errorlist nonfield">

<li>Generic validation error</li>

</ul>

# Looping over the form’s fields - - -

If you’re using the same HTML for each of your form fields, you can reduce duplicate code by looping through

each field in turn using a {% for %} loop:

{% for field in form %}

<div class="fieldWrapper">

{{ field.errors }}

{{ field.label\_tag }} {{ field }}

{% if field.help\_text %}

<p class="help" id="{{ field.auto\_id }}\_helptext">

{{ field.help\_text|safe }}

</p>

{% endif %}

</div>

{% endfor %}

Useful attributes on **{{ field }}** include:**{{ field.errors }}**

Outputs a **<ul class="errorlist">** containing any validation errors corresponding to this field. You

can customize the presentation of the errors with a **{% for error in field.errors %}** loop. In this

case, each object in the loop is a string containing the error message.

**{{ field.field }}**

The Field instance from the form class that this BoundField wraps. You can use it to access Field

attributes, e.g. **{{ char\_field.field.max\_length }}.**

**{{ field.help\_text }}**

Any help text that has been associated with the field.

**{{ field.html\_name }}**

The name of the field that will be used in the input element’s name field. This takes the form prefix

into account, if it has been set.

**{{ field.id\_for\_label }}**

The ID that will be used for this field (**id\_email** in the example above). If you are constructing the

label manually, you may want to use this in lieu of **label\_tag**. It’s also useful, for example, if you

have some inline JavaScript and want to avoid hardcoding the field’s ID.

**{{ field.is\_hidden }}**

This attribute is True if the form field is a hidden field and False otherwise. It’s not particularly useful

as a template variable, but could be useful in conditional tests such as: {% if field.is\_hidden %}

{# Do something special #}

{% endif %}

**{{ field.label }}**

The label of the field, e.g. Email address.

**{{ field.label\_tag }}**

The field’s label wrapped in the appropriate HTML <label> tag. This includes the form’s

label\_suffix. For example, the default label\_suffix is a colon:

<label for="id\_email">Email address:</label>

**{{ field.legend\_tag }}**

Similar to **field.label\_tag** but uses a **<legend>** tag in place of **<label>,** for widgets with multiple

inputs wrapped in a **<fieldset>.**

**{{ field.use\_fieldset }}**

This attribute is True if the form field’s widget contains multiple inputs that should be semantically

grouped in a **<fieldset>** with a **<legend>** to improve accessibility. An example uses in a template:

**{{ field.use\_fieldset }}**

This attribute is True if the form field’s widget contains multiple inputs that should be semantically

grouped in a <fieldset> with a <legend> to improve accessibility. An example uses in a template:

{% if field.use\_fieldset %}

<fieldset>

{% if field.label %}{{ field.legend\_tag }}{% endif %}

{% else %}

{% if field.label %}{{ field.label\_tag }}{% endif %}

{% endif %}

{{ field }}

{% if field.use\_fieldset %}</fieldset>{% endif %}

**{{ field.value }}**

The value of the field. e.g. **someone@example.com.**

Looping over hidden and visible fields - -

If you’re manually laying out a form in a template, as opposed to relying on Django’s default form layout, you

might want to treat <input type="hidden"> fields differently from non-hidden fields. For example, because

hidden fields don’t display anything, putting error messages “next to” the field could cause confusion for your

users – so errors for those fields should be handled differently.

Django provides two methods on a form that allow you to loop over the hidden and visible fields independently: hidden\_fields() and visible\_fields(). Here’s a modification of an earlier example that uses

these two methods:

{# Include the hidden fields #}

{% for hidden in form.hidden\_fields %}

{{ hidden }}

{% endfor %}

{# Include the visible fields #}

{% for field in form.visible\_fields %}

<div class="fieldWrapper">

{{ field.errors }}

{{ field.label\_tag }} {{ field }}

</div>

{% endfor %}

This example does not handle any errors in the hidden fields. Usually, an error in a hidden field is a sign

of form tampering, since normal form interaction won’t alter them. However, you could easily insert some

error displays for those form errors, as well.

--------- END ---------