Format localization

Overview

Django's formatting system is capable of displaying dates, times and numbers in templates using the format specified for the current locale. It also handles localized input in forms.

Two users accessing the same content may see dates, times and numbers formatted in different ways, depending on the formats for their current locale.

Note: To enable number formatting with thousand separators, it is necessary to set `USE_THOUSAND_SEPARATOR = True` in your settings file. Alternatively, you could use `intcomma` to format numbers in your template.

Note: There is a related `USE_I18N` setting that controls if Django should activate translation. See Translation for more details.

Locale aware input in forms

When formatting is enabled, Django can use localized formats when parsing dates, times and numbers in forms. That means it tries different formats for different locales when guessing the format used by the user when inputting data on forms.

Note: Django uses different formats for displaying data to those it uses for parsing data. Most notably, the formats for parsing dates can't use the `%a` (abbreviated weekday name), `%A` (full weekday name), `%b` (abbreviated month name), `%B` (full month name), or `%p` (AM/PM).

To enable a form field to localize input and output data use its `localize` argument:

```
class CashRegisterForm(forms.Form):
    product = forms.CharField()
    revenue = forms.DecimalField(max_digits=4, decimal_places=2, localize=True)
```

Controlling localization in templates

Django tries to use a locale specific format whenever it outputs a value in a template.

However, it may not always be appropriate to use localized values – for example, if you're outputting JavaScript or XML that is designed to be machine-readable, you will always want unlocalized values. You may also want to use localization in selected templates, rather than using localization everywhere.

To allow for fine control over the use of localization, Django provides the `l10n` template library that contains the following tags and filters.

Template tags

`localize`

Enables or disables localization of template variables in the contained block.

To activate or deactivate localization for a template block, use:

```
{% load l10n %}

{% localize on %}
    {{ value }}
{% endlocalize %}

{% localize off %}
    {{ value }}
{% endlocalize %}
```

When localization is disabled, the localization settings formats are applied.

See `localize` and `unlocalize` for template filters that will do the same job on a per-variable basis.

Template filters

`localize`

Forces localization of a single value.

For example:

```
{% load l10n %}

{{ value|localize }}
```

To disable localization on a single value, use `unlocalize`. To control localization over a large section of a template, use the `localize` template tag.

`unlocalize`

Forces a single value to be printed without localization.

For example:

```
{% load l10n %}

{{ value|unlocalize }}
```

To force localization of a single value, use `localize`. To control localization over a large section of a template, use the `localize` template tag.

Returns a string representation for numbers (`int` , `float` , or `Decimal`) with the localization settings formats applied.

Creating custom format files

Django provides format definitions for many locales, but sometimes you might want to create your own, because a format file doesn't exist for your locale, or because you want to overwrite some of the values.

To use custom formats, specify the path where you'll place format files first. To do that, set your `FORMAT_MODULE_PATH` setting to the package where format files will exist, for instance:

```
FORMAT_MODULE_PATH = [
    "mysite.formats",
    "some_app.formats",
]
```

Files are not placed directly in this directory, but in a directory named as the locale, and must be named `formats.py` . Be careful not to put sensitive information in these files as values inside can be exposed if you pass the string to `django.utils.formats.get_format()` (used by the `date` template filter).

To customize the English formats, a structure like this would be needed:

```
mysite/
    formats/
        __init__.py
        en/
            __init__.py
            formats.py
```

where `formats.py` contains custom format definitions. For example:

```
THOUSAND_SEPARATOR = "\xa0"
```

to use a non-breaking space (Unicode `00A0`) as a thousand separator, instead of the default for English, a comma.

Limitations of the provided locale formats

Some locales use context-sensitive formats for numbers, which Django's localization system cannot handle automatically.

Switzerland (German)

The Swiss number formatting depends on the type of number that is being formatted. For monetary values, a comma is used as the thousand separator and a decimal point for the decimal separator. For all other numbers, a comma is used as decimal separator and a space as thousand separator. The locale format provided by Django uses the generic separators, a comma for decimal and a space for thousand separators.