Django Leaflet Documentation

Release 0.20

Makina Corpus

May 05, 2017

Contents

Contents:

CHAPTER 1

Installation

Last stable version:

```
pip install django-leaflet
```

Last development version (master branch):

```
pip install -e git+https://github.com/makinacorpus/django-leaflet.git#egg=django-
↪leaflet
```

Configuration

- Add `leaflet` to your `INSTALLED_APPS`
- Add the HTML header:

```
{% load leaflet_tags %}

<head>
    ...
    {% leaflet_js %}
    {% leaflet_css %}
</head>
```

- Add the map in your page, providing a name:

```
...
<body>
    ...
    {% leaflet_map "yourmap" %}
    ...
</body>
```

- Your map shows up!

Example

Check out the example project for a complete integration!

CHAPTER 2

Use in templates

Use Leaflet API

You can use the *Leaflet* API as usual. There are two ways to grab a reference on the just initialized map and options.

Using Javascript callback function

The easy way :

```
<script type="text/javascript">
    function map_init_basic (map, options) {
        ...
        L.marker([50.5, 30.5]).addTo(map);
        ...
    }
</script>

{% leaflet_map "yourmap" callback="window.map_init_basic" %}
```

Using events

If you don't want to expose global callbacks :

```
<script type="text/javascript">
    window.addEventListener("map:init", function (e) {
        var detail = e.detail;
        ...
        L.marker([50.5, 30.5]).addTo(detail.map);
        ...
    }, false);
</script>
```

Event object has two properties : `map` and `options` (initialization).

For Internet Explorer support, we fallback on jQuery if available

```
$(window).on('map:init', function (e) {
    var detail = e.originalEvent ?
                 e.originalEvent.detail : e.detail;
    ...
    L.marker([50.5, 30.5]).addTo(detail.map);
    ...
});
```

If you want to support archaic browsers **and** still avoid jQuery, *django-leaflet* comes with a minimalist polyfill for events. Add it in <head> this way

```
<!--[if IE 8]><script src="{% static "leaflet/eventlister.ie8.js" %}"></script><!--<!
↪[endif]-->
<!--[if lt IE 8]><script src="{% static "leaflet/eventlister.ie6-7.js" %}"></script><!
↪--<![endif]-->
```

Customize map size

CSS is your friend:

```
<style>

    .leaflet-container {  /* all maps */
        width:  600px;
        height: 400px;
    }

    #specialbigmap {
        height: 800px;
    }

</style>
```

Configuration

In order to configure *django-leaflet*, just add a new section in your settings:

```
LEAFLET_CONFIG = {
    # conf here
}
```

And add some of the following entries.

Spatial extent

You can configure a global spatial extent for your maps, that will automatically center your maps, restrict panning and add reset view and scale controls. (*See advanced usage to tweak that.*):

```
'SPATIAL_EXTENT': (5.0, 44.0, 7.5, 46)
```

Initial map center and zoom level

In addition to limiting your maps with `SPATIAL_EXTENT`, you can also specify initial map center, default, min and max zoom level:

```
'DEFAULT_CENTER': (6.0, 45.0),
'DEFAULT_ZOOM': 16,
'MIN_ZOOM': 3,
'MAX_ZOOM': 18,
```

The tuple/list must contain (lat,lng) coords.

Default tiles layer

To globally add a tiles layer to your maps:

```
'TILES': 'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png'
```

This setting can also be a list of tuples `(name, url, options)`. The python dict `options` accepts all the Leaflet tileLayers options.

If it contains several layers, a layer switcher will then be added automatically.

```
'TILES': [('Satellite', 'http://server/a/...', {'attribution': '&copy; Big eye',
↪'maxZoom': 16}),
          ('Streets', 'http://server/b/...', {'attribution': '&copy; Contributors'})]
```

If you omit this setting, a default OpenSTreetMap layer will be created for your convenience. If you do not want a default layers (perhaps to add them in your own JavaScript code on map initialization), set the value to an empty list, as shown below.

```
'TILES': []
```

Note that this will also prevent any overlays defined in settings from being displayed.

Overlay layers

To globally add an overlay layer, use the same syntax as tiles:

```
'OVERLAYS': [('Cadastral', 'http://server/a/{z}/{x}/{y}.png', {'attribution': '&copy;␣
↪IGN'})]
```

Currently, overlay layers from settings are limited to tiles. For vectorial overlays, you will have to add them via JavaScript (see events).

Attribution prefix

To globally add an attribution prefix on maps (most likely an empty string)

```
'ATTRIBUTION_PREFIX': 'Powered by django-leaflet'
```

Default is `None`, which leaves the value to Leaflet's default.

Scale control

Scale control may be set to show 'metric' (m/km), or 'imperial' (mi/ft) scale lines, or 'both'. Default is 'metric'.

Enable metric and imperial scale control:

```
'SCALE': 'both'
```

Disable scale control:

```
'SCALE': None
```

Minimap control

Shows a small map in the corner which shows the same as the main map with a set zoom offset:

```
'MINIMAP': True
```

By default it shows the tiles of the first layer in the list.

(More info...)

Reset view button

By default, a button appears below the zoom controls and, when clicked, shows the entire map. To remove this button, set:

```
'RESET_VIEW': False
```

Global initialization functions and `window.maps`

Since 0.7.0, the `leaflet_map` template tag no longer registers initialization functions in global scope, and no longer adds map objects into `window.maps` array by default. To restore these features, use:

```
'NO_GLOBALS' = False
```

Force Leaflet image path

If you are using staticfiles compression libraries such as django_compressor, which can do any of compressing, concatenating or renaming javascript files, this may break Leaflet's own ability to determine its installed path, and in turn break the method `L.Icon.Default.imagePath()`.

To use Django's own knowledge of its static files to force this value explicitly, use:

```
'FORCE_IMAGE_PATH': True
```

Plugins

To ease the usage of plugins, django-leaflet allows specifying a set of plugins, that can later be referred to from the template tags by name:

```
'PLUGINS': {
    'name-of-plugin': {
        'css': ['relative/path/to/stylesheet.css', '/root/path/to/stylesheet.css'],
        'js': 'http://absolute-url.example.com/path/to/script.js',
        'auto-include': True,
    },
    . . .
}
```

Both 'css' and 'js' support identical features for specifying resource URLs:

- can be either a plain string or a list of URLs
- each string can be:
  - absolute URL - will be included as-is; **example**: `http://absolute-url.example.com/path/to/script.js`
  - a URL beginning from the root - will be included as-is; **example**: `/root/path/to/stylesheet.css`
  - a relative URL - settings.STATIC_URL will be prepended; **example**: `relative/path/to/stylesheet.css` will be included as **/static/relative/path/to/stylesheet.css** (depending on your setting for STATIC_URL)

Now, use `leaflet_js` and `leaflet_css` tags to load CSS and JS resources of configured Leaflet plugins.

By default only plugins with `'auto-include'` as True will be included.

To include specific plugins in the page, specify plugin names, comma separated:

```
{% load leaflet_tags %}

<head>
    ...
    {% leaflet_js  plugins="bouncemarker,draw" %}
    {% leaflet_css plugins="bouncemarker,draw" %}
</head>
```

To include all plugins configured in `LEAFLET_CONFIG['PLUGINS']`, use:

```
{% leaflet_js plugins="ALL" %}
{% leaflet_css plugins="ALL" %}
```

CHAPTER 3

Leaflet map forms widgets

A Leaflet widget is provided to edit geometry fields. It embeds *Leaflet.draw* in version *0.4.0*.

In Adminsite

```
from django.contrib import admin
from leaflet.admin import LeafletGeoAdmin

from .models import WeatherStation


admin.site.register(WeatherStation, LeafletGeoAdmin)
```

A mixin is also available for inline forms:

```
from django.contrib import admin
from leaflet.admin import LeafletGeoAdminMixin

class PoiLocationInline(LeafletGeoAdminMixin, admin.StackedInline):
    model = PoiLocation
```

In forms

With *Django* >= 1.6:

```
from django import forms

from leaflet.forms.widgets import LeafletWidget


class WeatherStationForm(forms.ModelForm):
```

```
    class Meta:
        model = WeatherStation
        fields = ('name', 'geom')
        widgets = {'geom': LeafletWidget()}
```

With all *Django* versions:

```
from django import forms

from leaflet.forms.fields import PointField


class WeatherStationForm(forms.ModelForm):
    geom = PointField()

    class Meta:
        model = WeatherStation
        fields = ('name', 'geom')
```

The related template would look like this:

```
{% load leaflet_tags %}
<html>
  <head>
   {% leaflet_js plugins="forms" %}
   {% leaflet_css plugins="forms" %}
  </head>
  <body>
    <h1>Edit {{ object }}</h1>
    <form action="POST">
        {{ form }}
        <input type="submit"/>
    </form>
  </body>
</html>
```

Every map field will trigger an event you can use to add your custom machinery :

```
map.on('map:loadfield', function (e) {
    ...
    // Customize map for field
    console.log(e.field, e.fieldid);
    ...
});
```

If you need a reusable customization of widgets maps, first override the JavaScript field behaviour by extending `L.GeometryField`, then in Django subclass the `LeafletWidget` to specify the custom `geometry_field_class`.

```
YourGeometryField = L.GeometryField.extend({
    addTo: function (map) {
        L.GeometryField.prototype.addTo.call(this, map);
        // Customize map for field
        console.log(this);
    },
    // See GeometryField source (static/leaflet/leaflet.forms.js) to override more␣
↪stuff...
```

```
});
```

```
class YourMapWidget(LeafletWidget):
    geometry_field_class = 'YourGeometryField'

class YourForm(forms.ModelForm):
    class Meta:
        model = YourModel
        fields = ('name', 'geom')
        widgets = {'geom': YourMapWidget()}
```

Plugins

It's possible to add extras JS/CSS or auto-include *forms* plugins everywhere:

```
LEAFLET_CONFIG = {
    'PLUGINS': {
        'forms': {
            'auto-include': True
        }
    }
}
```

(*It will be merged over default minimal set required for edition*)

Details

- It relies on global settings for map initialization.
- It works with local map projections. But SRID is specified globally through `LEAFLET_CONFIG['SRID']` as described below.
- Javascript component for de/serializing fields value is pluggable.
- Javascript component for Leaflet.draw behaviour initialization is pluggable.

CHAPTER 4

Advanced usage

`{% leaflet_map %}` tag parameters

- `callback`: javascript function name for initialization callback. (Default: None).
- `fitextent`: control if map initial view shoud be set to extent setting. (Default: `True`). Setting fixextent to `False` will prevent view reset and scale controls to be added.
- `creatediv`: control if the leaflet map tags creates a new div or not. (Default: `True`). Useful to put the javascript code in the header or footer instead of the body of the html document. If used, do not forget to create the div manually.
- `loadevent`: One or more space-separated *window* events that trigger map initialization. (Default: `load`, i.e. all page resources loaded). If empty values is provided, then map initialization is immediate. And with a wrong value, the map is never initialized. :)
- `settings_overrides`: Map with overrides to the default LEAFLET_CONFIG settings. (Default: {}).

Config overrides

It is possible to dynamically override settings in `LeafletWidget` init:

```
from leaflet.forms.widgets import LeafletWidget


class WeatherStationForm(forms.ModelForm):

    class Meta:
        model = WeatherStation
        fields = ('name', 'geom')
        widgets = {'geom': LeafletWidget(attrs={
            'settings_overrides': {
                'DEFAULT_CENTER': (6.0, 45.0),
```

```
            }
        })}
```

For overriding the settings in `LeafletGeoAdmin`, use set the appropriate property:

```
class WeatherStationAdminAdmin(LeafletGeoAdmin):
    settings_overrides = {
       'DEFAULT_CENTER': (6.0, 45.0),
    }
```

Projection

It is possible to setup the map spatial reference in `LEAFLET_CONFIG`:

```
'SRID': 2154  # See http://spatialreference.org
```

Additional parameter is required to compute scale levels : the tiles extent in local projection:

```
'TILES_EXTENT': [924861,6375196,985649,6448688],
```

For more information, have a look at this example.

Example of TileCache configuration compatible with Leaflet:

```
[scan-portrait]
type=WMSLayer
layers=scan100,scan25
url=http://server/wms?
extension=jpg
tms_type=google
srs=EPSG:2154
bbox=924861,6375196,985649,6448688

[cache]
type=GoogleDisk
expire=2592000
base=/tmp/tiles
```

By default, *django-leaflet* will try to load the spatial reference from your static files at "proj4js/{{ srid }}.js". If it fails, it will eventually rely on spatialreference.org.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search