Django Leaflet Documentation

Release 0.20

Makina Corpus

Jun 19, 2023

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
```

1.1 Dependencies

django-leaflet requires the GDAL library installed on the system. Installation instructions are platform-specific.

1.2 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!

1.3 Example

Check out the example project for a complete integration!

CHAPTER 2

Use in templates

2.1 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>
    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>
    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);
    ...
});
```

2.2 Customize map size

CSS is your friend:

```
<style>

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

    #specialbigmap {
        height: 800px;
    }

    /* Resize the "display_raw" textbox */
    .django-leaflet-raw-textarea {
        width: 100%;
    }

</style>
```

2.3 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.

2.3.1 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)
```

2.3.2 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, coordinate values precision:

```
'DEFAULT_CENTER': (6.0, 45.0),
'DEFAULT_ZOOM': 16,
'MIN_ZOOM': 3,
'MAX_ZOOM': 18,
'DEFAULT_PRECISION': 6,
```

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

Notice that if you do not use `SPATIAL_EXTENT`, but you do specify a value for the `DEFAULT_ZOOM` entry, you **must** also indicate a `DEFAULT_CENTER`, or you will get an empty, not working pane, for all new database records without a default geometry value. This is regarded as an error in the configuration, not a bug in the software.

2.3.3 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.

2.3.4 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 also events).

To add layers other than the tiles supported by the global config, e.g. WMS layers, insert a script block, get a reference to the map's `layerscontrol`, and add any layer supported by Leaflet as overlays to that layerscontrol object.

In a template:

```
{% block content %}
{% leaflet_map "detailmap" callback="window.map_init" %}
{% endblock %}


{% block javascript %}
{{ block.super }}
<script>
function map_init(map, options) {
    {% include 'shared/overlays.html' %}
}
</script>
{% endblock %}
```

In a snippet, here called `shared/overlays.html`, the overlays are configured. Doing so in a snippet allows the same set of overlays to be re-used across other maps in your Django project.

`shared/overlays.html`:

```
var lc = map.layerscontrol;

// An example from the Atlas of Living Australia https://www.ala.org.au/
lc.addOverlay(
    L.tileLayer.wms(
    'https://spatial-beta.ala.org.au/geoserver/ALA/wms',
    {layers: 'ALA:aus2', format: 'image/png', transparent: true}),
    'Australia'
);

// add lc.addOverlay() layers as needed
```

For an overview of available layer types and options, see the [Leaflet docs on tile layers](https://leafletjs.com/examples/wms/wms.html).

2.3.5 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.

2.3.6 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
```

2.3.7 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...)

2.3.8 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
```

2.3.9 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
```

2.3.10 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
```

2.3.11 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>
```

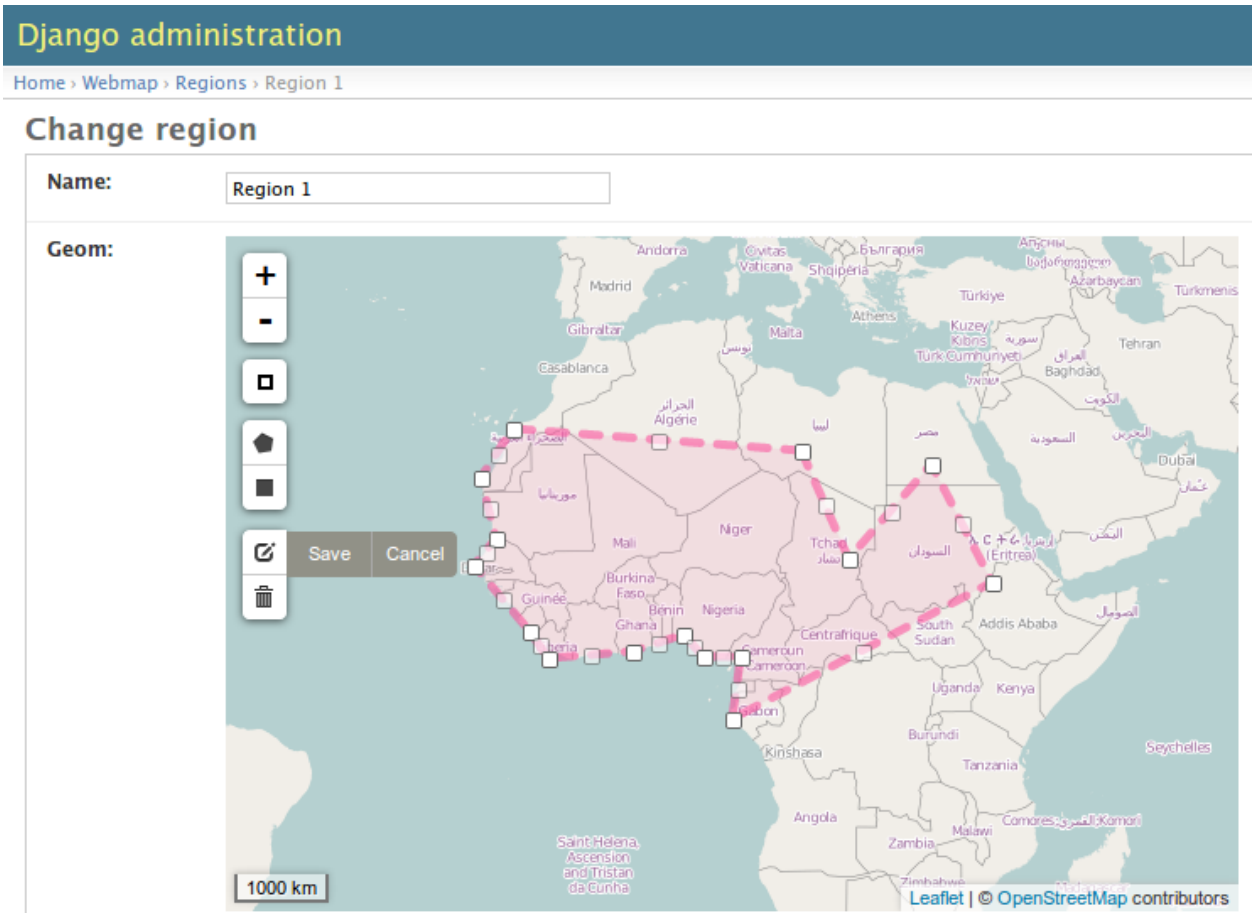
To include all plugins configured in `LEAFLET_CONFIG['PLUGINS']`, use:

```
{% 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*.



3.1 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
```

To modify the map widget used in the Django admin, override a custom `admin/change_form.html`:

```
{% extends "admin/change_form.html" %}
{% load i18n admin_urls static leaflet_tags %}

{% block stylesheets %}
{{ block.super }}
{% leaflet_css plugins="ALL" %}
<style>
/* Force leaflet controls underneath header (z-index 1000) and
   above leaflet tiles (z-index 400)*/
.leaflet-top{z-index:999;}
</style>
{% endblock %}

{% block javascripts %}
{{ block.super }}
{% leaflet_js plugins="ALL" %}
{% include 'shared/leaflet_widget_overlays.js' %}
{% endblock %}
```

Note: Django 3.2 admin/change_form.html renamed `stylesheet` block to `extrastyle` and `javascripts` to `admin_change_form_document_ready`, adjust your template accordingly.

In this way, both CSS and JS can be modified for all admin leaflet widgets.

As an example of modifying the CSS, here the leaflet map widget controls are forced underneath a bootstrap4 navbar.

As an example of modifying the JS, a custom snippet called `shared/leaflet_widget_overlays.js` uses the map init event to add some custom (non-tile) overlays.

```
<script>
  window.addEventListener("map:init", function (event) {
    var map = event.detail.map; // Get reference to map
    {% include 'shared/overlays.html' %}

    // Other modifications, e.g. fullscreen control:
    map.addControl(new L.Control.Fullscreen());
    // Note, this requires the Leaflet fullscreen CSS, JS,
    // and image assets to be present as static files,
```

(continues on next page)

(continued from previous page)

```
    // and configured in LEAFLET_SETTINGS
});
</script>
```

Again, the actual overlays here are factored out into a separate snippet. In this example, we re-use `shared/overlays.html` as also shown in *Overlay layers*.

To show a textarea input for the raw GeoJSON geometry, override admin `form_fields`:

```
from django.contrib.gis.db import models as geo_models

LEAFLET_WIDGET_ATTRS = {
    'map_height': '500px',
    'map_width': '100%',
    'display_raw': 'true',
    'map_srid': 4326,
}

LEAFLET_FIELD_OPTIONS = {'widget': LeafletWidget(attrs=LEAFLET_WIDGET_ATTRS)}

FORMFIELD_OVERRIDES = {
    geo_models.PointField: LEAFLET_FIELD_OPTIONS,
    geo_models.MultiPointField: LEAFLET_FIELD_OPTIONS,
    geo_models.LineStringField: LEAFLET_FIELD_OPTIONS,
    geo_models.MultiLineStringField: LEAFLET_FIELD_OPTIONS,
    geo_models.PolygonField: LEAFLET_FIELD_OPTIONS,
    geo_models.MultiPolygonField: LEAFLET_FIELD_OPTIONS,
}

class MyAdmin(admin.ModelAdmin):

    formfield_overrides = FORMFIELD_OVERRIDES
```

The widget attribute *display_raw* toggles the textarea input. The textarea can be resized by overriding its CSS class `.django-leaflet-raw-textarea`.

3.2 In forms

```
from django import forms

from leaflet.forms.widgets import LeafletWidget


class WeatherStationForm(forms.ModelForm):

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

Again, the LeafletWidget can be intialized with custom attributes, e.g. `LeafletWidget(attrs=LEAFLET_WIDGET_ATTRS)` as shown above.

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 method="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);
    ...
});
```

3.2.1 Programmatically appended maps

If you are adding a map to the DOM programmatically, as for example by jQuery, the default events driven mechanism will not work, and a viable workaround is to specify an empty `loadevent` attribute in your `Meta.widgets` definiton :

```
class Meta:
    ...
    widgets = {
        'geometry': LeafletWidget(attrs={'loadevent': ''}),
    }
```

You will also need to refresh the map by invoking `invalidateSize` on it, and to do so you need to instruct django-leaflet to expose the map globally, by setting the `NO_GLOBALS` to False, in `LEAFLET_CONFIG`. The map will be accessible via a field added to the global `window` object: if `xyzt` is the name of your field, your corresponding leaflet map will be at `window['leafletmapid_xyzt-map']`.

3.2.2 Custom Forms

If you need a reusable customization of widgets maps, first override the JavaScript field behavior 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);
    },
```

(continues on next page)

(continued from previous page)

```
    // 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()}
```

To customise individual forms, you can either extend the geometry field as shown above, or inject a script into the form template.

In this example, a custom set of overlays is added as shown for both *Overlay layers* and *In Adminsite* widgets, insert an extra script into the form template in the same way as shown in *In Adminsite*.

```
{% extends "base.html" %}
{% load static leaflet_tags geojson_tags crispy_forms_tags bootstrap4  %}

<!-- The form -->
{% block content %}
<div class="container">
  <div class="row">
    <div class="col-12">
      {% crispy form form.helper %}
    </div><!-- .col -->
  </div><!-- .row -->
</div><!-- .container -->
{% endblock %}

{% block extrastyle %}
{% leaflet_css plugins="ALL" %}
{{ form.media.css }}
{% endblock %}

{% block extrajs %}
{% leaflet_js plugins="ALL" %}
{{ form.media.js }}
{% include 'shared/leaflet_widget_overlays.js' %}
{% endblock extrajs %}
```

3.3 Plugins

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

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

(continues on next page)

(continued from previous page)

```
    }
}
```

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

3.4 Customizing serialization

Leaflet fields are easily customizable by setting `field_store_class` and `geometry_field_class`. For example, if we have a field with three dimensions and we want to add the third dimension as 0 by default, we could use the following widget:

```
class Leaflet3dWidget(LeafletWidget):
    field_store_class = 'L.Field3dStore'

    class Media:
        js = ['leaflet/leaflet.forms.js', 'leaflet.3d.js']
```

With the following `leaflet.3d.js`:

```
L.Field3dStore = L.FieldStore.extend({
    _serialize: function (layer) {
        const geojson = JSON.parse(L.FieldStore.prototype._serialize.call(this,␣
↪layer));
        for (const coordinate of geojson.coordinates) {
            coordinate.push(0);
        }
        return JSON.stringify(geojson);
    }
});
```

For more information on what is customizable, please see the `leaflet/leaflet.forms.js`.

3.5 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

4.1 {% 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: {}).

4.2 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),
```

(continues on next page)

(continued from previous page)

```
            }
        })}
```

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

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

4.3 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