olwidget Documentation

Release 0.3

Charlie DeTar

Sep 27, 2017

Contents

This is documentation for an old version of olwidget, v0.3. You can get the current version here.

CHAPTER 1

django-olwidget documentation

Installation

Installation of the `olwidget` Django app requires a couple of steps – these should be familiar to users of other reusable Django apps:

1. Install `django-olwidget` using your preferred method for python modules. The folder `olwidget` in the `django-olwidget` directory must end up in your python path. You could do this by copying or symlinking it into your path, or with:

   ```
   python setup.py install
   ```

   The preferred method is to install from git source using pip with virtualenv. The requirements file entry for this is as follows:

   ```
   -e git://github.com/yourcelf/olwidget.git#egg=django-olwidget
   ```

2. Copy or link the `media/olwidget` directory into to your project's `MEDIA_ROOT`. If you wish to name the directory something other than `olwidget`, define `OLWIDGET_MEDIA_URL` with the URL for the media files in your settings file.
3. Include `'olwidget'` in your project's settings `INSTALLED_APPS` list.
4. (Optional) If you want to use Google, Yahoo or CloudMade map layers, you must include `GOOGLE_API_KEY`, `YAHOO_APP_ID` or `CLOUDMADE_API_KEY` in your settings file. `olwidget` uses an OpenStreetMaps layer by default, which requires no key.

Map widgets

`olwidget` defines 3 widget types that can be used to display maps:

EditableMap

`EditableMap` is a widget type for use in forms to allows editing of geometries. Constructor:

```
olwidget.widgets.EditableMap(options=None, template=None)
```

options A dict of options for map display. See *Options* below.

template A template to use for rendering the map.

An example form definition that uses an editable map:

```
from django import forms
from olwidget.widgets import EditableMap

class MyForm(forms.Form):
    location = forms.CharField(widget=EditableMap())
```

In a template:

```
<head> {{ form.media }} </head>
<body>... {{ form }} ...</body>
```

MapDisplay

`MapDisplay` is used for displaying geometries on read-only maps. Constructor:

```
olwidget.widgets.MapDisplay(fields, options=None, template=None)
```

fields A list of geometries to display on the map, either valid geometry strings or geometry fields.

options A dict of options for map display. See *Options* below.

template A template to use for rendering the map.

An example simple read-only display map:

```
from olwidget.widgets import MapDisplay

map = MapDisplay(fields=[mymodel.start_point, mymodel.destination])
```

In a template:

```
<head> {{ map.media }} </head>
<body>... {{ map }} ...</body>
```

InfoMap

`InfoMap` is used for displaying read-only maps with clickable information popups over geometries. Constructor:

```
olwidget.widgets.InfoMap(info, options=None, template=None)
```

info A list of `[geometry, attr]` pairs. `attr` can be either a string containing html, or a dict containing `html` and `style` keys. The html is displayed when the geometry is clicked.

options A dict of options for map display. See *Options* below.

template A template to use for rendering the map.

An example info map:

```
from olwidget.widgets import InfoMap

map = InfoMap([
    [mymodel.point, "<p>This is where I had my first kiss.</p>"],
    [othermodel.polygon, "<p>This is my home town.</p>"],
    [othermodel.point, {
        'html': "<p>Special style for this point.</p>",
        'style': {'fill_color': '#00FF00'},
    }],
    ...
])
```

In a template:

```
<head> {{ map.media }} </head>
<body>... {{ map }} ...</body>
```

Inside Admin

`olwidget` has several advantages over the built-in geodjango admin map implementation, including greater map customization, support for more geometry types, and the option to include a map in admin changelist pages.

To use `olwidget` for admin, simply use `olwidget.admin.GeoModelAdmin` or a subclass of it as the ModelAdmin type for your model.

Example using `olwidget` in admin:

```
# admin.py

from django.contrib import admin
from olwidget.admin import GeoModelAdmin
from myapp import Restaurant, Owner

# Use the default map
admin.site.register(Restaurant, GeoModelAdmin)

# Customize the map
class MyGeoAdmin(GeoModelAdmin):
    options = {
        'layers': ['google.streets'],
        'default_lat': 44,
        'default_lon': -72,
    }

admin.site.register(Owner, MyGeoAdmin)
```

`olwidget.admin.GeoModelAdmin` uses the `olwidget.widgets.EditableMap` for display, so all the map display options listed below in *Options* for editable map types can be used with maps in admin.

Changelist maps

To show a clickable map on the admin changelist page, use the `list_map` property to specify which fields to display:

```
# an example model:

class City(models.Model):
    location = models.PointField()

# admin.py

from django.contrib import admin
from olwidget.admin import GeoModelAdmin
from myapp import City

class CityGeoAdmin(GeoModelAdmin):
    list_map = ['location']

admin.site.register(City, CityGeoAdmin)
```

Options can be set for the changelist map using the `list_map_options` property:

```
class CityGeoAdmin(GeoModelAdmin):
    list_map = ['location']
    list_map_options = {
        # group nearby points into clusters
        'cluster': True,
        'cluster_display': 'list',
    }
```

Changelist maps use the `olwidget.widgets.InfoMap` type for display, so all the options listed below in *Options* for InfoMap types can be used for `list_map_options`.

Options

All of the `olwidget` map types can be passed an `options` dictionary that controls the look and feel of the map. An example:

```
from olwidget.widgets import MapDisplay

map = MapDisplay(options={
    'layers': ['osm.mapnik', 'google.hybrid', 'yahoo'],
    'default_lat': 44,
    'default_lon': -72,
})
```

Common options

The following options are shared by all `olwidget` map types:

`name` (string; defaults to `"data"`) The name of the overlay layer for the map (shown in the layer switcher).

`layers` (list; default `['osm.mapnik']`) A list of map base layers to include. Choices include `'osm.mapnik'`, `'osm.osmarender'`, `'google.streets'`, `'google.physical'`, `'google.satellite'`, `'google.hybrid'`, `'ve.road'`, `'ve.shaded'`, `'ve.aerial'`, `'ve.hybrid'`, `'wms.map'`, `'wms.nasa'`, `'yahoo.map'`, and `'cloudmade.<num>'` (where `<num>` is the number for a cloudmade style). A blank map can be obtained using `'wms.blank'`.

`default_lat` (float; default 0) Latitude for the center point of the map.

default_lon (float; default 0) Longitude for the center point of the map.

default_zoom (int; default 4) The zoom level to use on the map.

zoom_to_data_extent (True/False; default True) If `True`, the map will zoom to the extent of its vector data instead of `default_zoom`, `default_lat`, and `default_lon`. If no vector data is present, the map will use the defaults.

overlay_style (dict) A dict of style definitions for the geometry overlays. For more on overlay styling, consult the OpenLayers styling documentation. Options include:

- `fill_color`: (string) HTML color value
- `fill_opacity`: (float) opacity of overlays from 0 to 1
- `stroke_color`: (string) HTML color value
- `stroke_opacity`: (float) opacity of strokes from 0 to 1
- `stroke_width`: (int) width in pixels of lines and borders
- `stroke_linecap`: (string) Default is `round`. Options are `butt`, `round`, `square`.
- `stroke_dash_style`: (string) Default is `solid`. Options are `dot`, `dash`, `dashdot`, `longdash`, `longdashdot`, `solid`.
- `cursor`: (string) Cursor to be used when mouse is over a feature. Default is an empty string.
- `point_radius`: (integer) radius of points in pixels
- `external_graphic`: (string) URL of external graphic to use in place of vector overlays
- `graphic_height`: (int) height in pixels of external graphic
- `graphic_width`: (int) width in pixels of external graphic
- `graphic_x_offset`: (int) x offset in pixels of external graphic
- `graphic_y_offset`: (int) y offset in pixels of external graphic
- `graphic_opacity`: (float) opacity of external graphic from 0 to 1.
- `graphic_name`: (string) Name of symbol to be used for a point mark.
- `display`: (string) Can be set to `none` to hide features from rendering.

overlay_style_context (dict) A dict containing javascript functions which expand symbolizers in `overlay_style`. See this example for a javascript usage example.

map_div_class (string; default '') A CSS class name to add to the div which is created to contain the map.

map_div_style (dict, default {width: '600px', height: '400px'}) A set of CSS style definitions to apply to the div which is created to contain the map.

map_options (dict) A dict containing options for the OpenLayers Map constructor. Properties may include:

- `units`: (string) default `'m'` (meters)
- `projection`: (string) default `"EPSG:900913"` (the projection used by Google, OSM, Yahoo, and VirtualEarth).
- `display_projection`: (string) default `"EPSG:4326"` (the latitude and longitude we're all familiar with).
- `max_resolution`: (float) default `156543.0339`. Value should be expressed in the projection specified in `projection`.

- `max_extent`: default `[-20037508.34, -20037508.34, 20037508.34, 20037508.34]`. Values should be expressed in the projection specified in `projection`.
- `controls`: (array of strings) default `['LayerSwitcher', 'Navigation', 'PanZoom', 'Attribution']` The strings should be class names for map controls, which will be instantiated without arguments.

Any additional parameters available to the OpenLayers.Map.Constructor may be included, and will be passed directly.

Additional options for `EditableMap` and `MapDisplay` types

In addition to the common options listed above, `EditableMap` and `MapDisplay`, and `GeoModelAdmin` accept the following options:

`hide_textarea` (boolean; default `true`) Hides the textarea if true.

`editable` (boolean, default `true`) If true, allows editing of geometries.

Additional options for `InfoMap`

In addition to the common options listed above, `InfoMap` accepts the following options:

`popups_outside` (boolean; default `false`) If false, popups are contained within the map's viewport. If true, popups may expand outside the map's viewport.

`popup_direction` (string; default `auto`) The direction from the clicked geometry that a popup will extend. This may be one of:

- `tr` – top right
- `tl` – top left
- `br` – bottom right
- `bl` – bottom left
- `auto` – automatically choose direction.

`cluster` (boolean; default `false`) If true, points will be clustered using the OpenLayers.Strategy.ClusterStrategy. (see this cluster example).

`cluster_display` (string; default `'paginate'`) The way HTML from clustered points is handled:

- `'list'` – constructs an unordered list of contents
- `'paginate'` – adds a pagination control to the popup to click through the different points' HTML.

`cluster_style` (dict) The default style is:

```
{
    point_radius: "${radius}",
    stroke_width: "${width}",
    label: "${label}",
    font_size: "11px",
    font_family: "Helvetica, sans-serif",
    font_color: "#ffffff"
}
```

The arguments expressed with `${}` are programmatically replaced with values based on the cluster. Setting them to specific values will prevent this programatic replacement.

CHAPTER 2

olwidget.js documentation

Introduction

`olwidget` is a javascript library to make editing and displaying geometries and information on OpenLayers maps easier. It defines two primary types:

- `olwidget.EditableMap` is a map class that turns a textarea containing WKT geometry into an editable map. The map writes EWKT geometry (WKT with explicit SRID) to the textarea for later processing in forms. This is useful for anywhere you need a form to enter or edit precise geographic geometries for storage in a GIS database such as PostGIS.
- `olwidget.InfoMap` is a map class that displays popup info-windows over geometries when clicked. The popups are stylable using CSS, work with clustered as well as non-clustered points, and can be shown both inside and outside the Map's "viewport".

`olwidget` supports multiple geometry types, any number of maps per page, multiple map providers, and much more (see *Common options* below). A simple example:

```
<html>
    <head>
        <script type='text/javascript' src='http://openlayers.org/api/2.8/OpenLayers.
↪js'></script>
        <script type='text/javascript' src='http://openstreetmap.org/openlayers/
↪OpenStreetMap.js'></script>
        <script type='text/javascript' src='js/olwidget.js'></script>
        <link rel='stylesheet' href='css/olwidget.css' />
    </head>
    <body>
        <!-- a map with an editable overlay -->
        <textarea id='my_point'>SRID=4326;POINT(0 0)</textarea>
        <script type='text/javascript'>
            new olwidget.EditableMap('my_point', {name: 'My Point'});
        </script>

        <!-- a map displaying an info popup over a geometry -->
```

```
        <div id='map'></div>
        <script type='text/javascript'>
            new olwidget.InfoMap('map', [
                ['SRID=4326;POINT(0 0)', "<p>Here at the zero point!</p>"]
            ]);
        </script>
    </body>
</html>
```

Installation

Copy or link the olwidget/js/, olwidget/css/, and olwidget/img/ directories into your website's path. The three directories should share a parent.

Examples

See *Examples* for several examples of olwidget in action.

Documentation

olwidget.EditableMap constructor

Format:

```
new olwidget.EditableMap(<textareaId>, [options]);
```

- `textareaId`: the DOM id of the textarea to replace
- `options`: An object containing options for the resulting map. All fields are optional.

In addition to the common *options* listed below, `EditableMap` accepts the following options:

`geometry` (Array or string; defaults to `'point'`) The geometry to use for this map. Choices are `'point'`, `'linestring'`, and `'polygon'`. To allow multiple geometries, use an array such as `['point', 'linestring', 'polygon']`.

`isCollection` (boolean, default `false`) If true, allows multiple points/lines/polygons.

`hideTextarea` (boolean; default `true`) Hides the textarea if true.

`editable` (boolean, default `true`) If true, allows editing of geometries.

olwidget.InfoMap constructor

Format:

```
new olwidget.InfoMap(<mapDivId>, <infoArray>, [options]);
```

- `mapDivId`: the DOM id of a div to replace with this map.
- `infoArray`: an Array of (E)WKT geometries and content HTML for popups, such as:

```
[
    ["SRID=4326;POINT(0 0)", "<p>This is the zero point.</p>"],
    ["SRID=4326;POINT(10 10)", "<p>This is longitude 10 and latitude 10.</p>"],
    ...
]
```

Geometries can be displayed with individual styles by passing an object containing `html` and `style` keys instead of an HTML string:

```
[
    ["SRID=4326;POINT(10 10)", {
        html: "<p>A good looking point</p>",
        style: {
            fillColor: '#00FF00'
        }
    }],
    ...
]
```

- `options`: An object containing options for the resulting map. All fields are optional.

In addition to the common *options* listed below, `InfoMap` accepts the following options:

`popupsOutside` (boolean; default `false`) If false, popups are contained within the map's viewport. If true, popups may expand outside the map's viewport.

`popupDirection` (string; default `auto`) The direction from the clicked geometry that a popup will extend. This may be one of:

- `tr` – top right
- `tl` – top left
- `br` – bottom right
- `bl` – bottom left
- `auto` – automatically choose direction.

`cluster` (boolean; default `false`) If true, points will be clustered using the OpenLayers.Strategy.ClusterStrategy. (see this cluster example).

`clusterDisplay` (string; default `'paginate'`) The way HTML from clustered points is handled:

- `'list'` – constructs an unordered list of contents
- `'paginate'` – adds a pagination control to the popup to click through the different points' HTML.

Common options

The following options are shared by `olwidget.EditableMap` and `olwidget.InfoMap`:

`name` (string; defaults to `"data"`) The name of the overlay layer for the map (shown in the layer switcher).

`layers` (Array; default `['osm.mapnik']`) A list of map base layers to include. Choices include `'osm.mapnik'`, `'osm.osmarender'`, `'google.streets'`, `'google.physical'`, `'google.satellite'`, `'google.hybrid'`, `'ve.road'`, `'ve.shaded'`, `'ve.aerial'`, `'ve.hybrid'`, `'wms.map'`, `'wms.nasa'`, `'yahoo.map'`, and `'cloudmade.<num>'` (where `<num>` is the number for a cloudmade style). A blank map can be obtained using `'wms.blank'`. Additional providers or options can be manually added using the normal OpenLayers apis (see this provider example).

You must also include whatever javascript sources are needed to use these (e.g. maps.google.com or openstreetmap.org apis). For CloudMade maps, use the included `cloudmade.js` file, and append the API key as the hash portion of the URL, for example:

```
<!-- API key for http://olwidget.org -->
<script src="js/cloudmade.js#06c005818e31487cb270b0bdfc71e115" type="text/
↪javascript"></script>
```

See the other providers for a full example of all built-in layer providers.

`defaultLat` (float; default 0) Latitude for the center point of the map. For `olwidget.EditableMap`, this is only used if there is no geometry (e.g. the textarea is empty).

`defaultLon` (float; default 0) Longitude for the center point of the map. For `olwidget.EditableMap`, this is only used if there is no geometry (e.g. the textarea is empty).

`defaultZoom` (int; default 4) The zoom level to use on the map. For `olwidget.EditableMap`, this is only used if there is no geometry (e.g. the textarea is empty).

`zoomToDataExtent` (boolean; default `true`) If `true`, the map will zoom to the extent of its vector data instead of `defaultZoom`, `defaultLat`, and `defaultLon`. If no vector data is present, the map will use the defaults.

`overlayStyle` (object) An object with style definitions for the geometry overlays. See OpenLayers styling.

`overlayStyleContext` (object) An object with functions which expand `'${var}'` symbolizers in style definitions. See OpenLayers Style context.

`mapDivClass` (string; default `''`) A CSS class name to add to the div which is created to contain the map.

`mapDivStyle` (object, default `{width: '600px', height: '400px'}`) A set of CSS style definitions to apply to the div which is created to contain the map.

`mapOptions` (object) An object containing options for the OpenLayers Map constructor. Properties may include:

- `units`: (string) default `'m'` (meters)
- `projection`: (string) default `"EPSG:900913"` (the projection used by Google, OSM, Yahoo, and VirtualEarth – See *Projections* below).
- `displayProjection`: (string) default `"EPSG:4326"` (the latitude and longitude we're all familiar with – See *Projections* below).
- `maxResolution`: (float) default `156543.0339`. Value should be expressed in the projection specified in `projection`.
- `maxExtent`: default `[-20037508.34, -20037508.34, 20037508.34, 20037508.34]`. Values should be expressed in the projection specified in `projection`.
- `controls`: (array of strings) default `['LayerSwitcher', 'Navigation', 'PanZoom', 'Attribution']` The strings should be class names for map controls, which will be instantiated without arguments.

Any additional parameters available to the OpenLayers.Map.Constructor may be included, and will be passed directly.

Projections

`olwidget` uses the projections given in `mapOptions` to determine the input and output of WKT data. By default, it expects incoming WKT data to use `"EPSG:4326"` (familiar latitudes and longitudes), which is transformed internally to the map projection (by default, `"EPSG:900913"`, the projection used by OpenStreetMaps, Google, and

others). Currently, `olwidget` ignores the SRID present in any initial WKT data, and uses the projection specified in `mapOptions.displayProjection` to read the data.

To change the projection used for WKT, define the `mapOptions.displayProjection`. For example, the following will use `EPSG:900913` for all WKT data in addition to map display:

```
new olwidget.EditableMap('textareaId', {
    mapOptions: {
        projection: "EPSG:900913",
        displayProjection: "EPSG:900913"
    }
});
```

CHAPTER 3

Examples

The following are examples of using `olwidget.js` in HTML pages. See *olwidget.js documentation* for standalone javascript documentation, and *django-olwidget documentation* for use in Django.

In addition, the source distribution contains an example project called `test_project` which illustrates use of the Django app.

Editable Maps

- Simple example
- Collections of multiple geometries.
- Read only maps and multiple maps per page
- Custom starting locations and colors
- Other map providers, including Google, Yahoo, Microsoft VE, OpenStreetMaps, MetaCarta, and CloudMade.
- Multiple geometry types
- Other projections

Info Maps

- Info popups over geometries
- Clustered points
- Popups inside and outside map viewport
- Info map with image markers
- Info map with different styles per geometry
- Info map with clustered points and per-geometry styles

CHAPTER 4

Backwards incompatible changes introduced in v0.3

django-olwidget changes

- 2009-04-13: Changed the media for the olwidget app to live in "olwidget/media/olwidget/..." by default, in accordance with the convention proposed by django-staticmedia (thanks to skyl for suggesting this good idea). This doesn't change any URLs, but it has the potential to break symlink paths from previous installations.

CHAPTER 5

Backwards incompatible changes introduced in v0.2

olwidget.js changes

- `olwidget.Map` has been renamed `olwidget.EditableMap`
- Many of the options parameters' names have changed, mostly to conform to OpenLayers' mixedCase standard:
  - `default_lat` is now `defaultLat`
  - `default_lon` is now `defaultLon`
  - `default_zoom` is now `defaultZoom`
  - `overlay_style` is now `overlayStyle`
  - `map_class` is now `mapDivClass` (note addition of "Div")
  - `map_style` is now `mapDivStyle` (note addition of "Div")
  - `map_options` is now `mapOptions`
  - `is_collection` is now `isCollection`
  - `hide_textarea` is now `hideTextarea`
  - `'yahoo'` map layer is now called `'yahoo.map'`

  All internal methods and variables have also changed to use mixedCase.
- Map types now inherit from `OpenLayers.Map`. This affects javascript customizations that access the base map type.

  The old way:

```
var mymap = new olwidget.Map('textareaId');
mymap.map.zoomTo(4);
```

  The new way:

```
var mymap = new olwidget.EditableMap('textareaId');
mymap.zoomTo(4);
```

Django app changes

- `olwidget.widgets.OLWidget` has been renamed `olwidget.widgets.EditableMap`
- The `"olwidget/olwidget.html"` template has been renamed `"olwidget/editable_map.html"`
- The `admin.custom_geo_admin` method has been removed. Instead, just subclass `olwidget.admin.GeoModelAdmin`.
- `olwidget.admin` No longer inherits from `django.contrib.admin`. The old way:

```
from olwidget import admin

# no longer works
admin.site.register(MyModel, admin.GeoModelAdmin)
```

Instead, import admin from `django.contrib` as normal, and import `GeoModelAdmin` from `olwidget`, like this:

```
from django.contrib import admin
from olwidget.admin import GeoModelAdmin

admin.site.register(MyModel, GeoModelAdmin)
```