Научна конференция „Иновационни софтуерни инструменти и технологии с приложения в научни изследвания по математика, информатика и педагогика на обучението“, 23-24 ноември 2017 г., Пампорово
Scientific Conference „Innovative Software Tools and Technologies with Applications in Research in Mathematics, Informatics and Pedagogy of Education”, 23-24 November 2017, Pamporovo, Bulgaria

JAVASCRIPT LIBRARY FOR GEOGRAPHIC MAPS VISUALIZATION

Teodora Gardzheva, Asen Rahnev, Nikolay Pavlov, Angel Golev

Abstract. The paper presents a JavaScript library for visualization of geographic maps in web-based applications, which are syntactically and functionally compatible with OpenLayers 2. The library avoids the difficulties caused by rapid evolution of popular mapping tools like Google Maps API and Bing Maps API. It supports OpenLayers maps as well as Google maps. The paper provides a description of library's object-orientated model and functionality. The usage of the library in end-client applications is described in the paper as well. The library is integrated into an existing application for management of autopark.

Keywords: Map layers, Point of interest, Abstraction

1. Introduction

The popularity of map tracking and displaying data on a map have grown rapidly in recent years. There are published API (Application programming interface) libraries that facilitate integration of map functionality in web-based applications. OpenLayers and Google Maps API are among the most popular ones.

Google Maps API is a Javascript API under Google license [1]. The API provides a library for the integration of Google Maps into web sites, web or mobile applications. Google Maps are available through Google Maps API only. The API provides a wide range of services and utilities for data visualization on a map.

OpenLayers is a Javascript API that provides free use of dynamic maps in any web page [2]. The API does not have its own map provider, therefore it works with external ones. Through wrappers the OpenLayers API can use almost any data source to display a map data. Google maps, OpenStreetMap, Bing and others are among the external map providers, that the API implements wrappers for. The

OpenLayers class library presents a rich variety of map components and features. It is a flexible, extensible library with strong open-source community. It has a lot of available plug-ins developed by the community: like vector editing features, multi-projections support, Web map service (WMS), Web feature service (WFS) and other Geographic information system (GIS) friendly APIs [3]. Still one of the most important OpenLayers functionality is the Google maps integration. According to [4] statistics Google maps takes over 50% of mapping usage on the internet.

OpenLayers 2 is the latest OpenLayers version that supports the usage of Google maps layers via integrated functionality. OpenLayers 2 was released for the first time on August 25, 2006. It was OpenLayers' official stable release until OpenLayers 3 first release. Meanwhile, the ability to use almost any data source on a map (including Google maps) gathered popularity to OpenLayers among the websites that use mapping tools. Unfortunately, according to the information published at OpenLayers 2 documentation [2]: “Although Openlayer 2 continues to work and is still in use, development work is concentrating on version 3“. At the same time Google Maps API is under rapid development. With the release of version 3.23 of Google maps API a lot of OpenLayers 2 clients faced a problem: Google map disappeared. Therefore, OpenLayers 2 clients that are depending on Google maps were unable to use their common functionality. The problem was caused by new implementations in Google Maps API that OpenLayers library did not support at this moment. A lot hot fixes had been published, but most of them caused another unusual behaviour on the map [5, 6].

This paper describes a solution to the mentioned problem. We developed a Javascript API for dynamic maps that is syntactically and functionally compatible with OpenLayers 2 interface. It is designed for integration in already working business applications. The main goal is to keep the usage of Google maps and the rich functionality of OpenLayers. The developed API allows the developer to keep their already existing code. At the same time it provides fully functional usage of Google Maps and OpenLayers 2 maps, working independently from each other. The API's implementation supports same features and components on Google and OpenLayers maps. The library integration is almost effortless.

The described library implements a bridge between OpenLayers API and Google Maps API. It uses native OpenLayers functionality and native Google Maps API functionality to manage map instances independently. The aim is to separate the two APIs. That way both of them can be used to display independent base maps. This allows developers to use the latest Google Maps API versions. Our library serves as a map abstraction layer that stands above OpenLayers and Google maps, and unifies the communication between the application functionality and actual map library. Our library manages map instantiations, event handling and release of unused objects. It supports dynamic switching between OpenLayers and Google maps.

OpenLayers provides a wide variety of features on a map. Custom components can be drawn on a map using OpenLayers markers and vector features. Analogously Google maps exposes a rich class library, but the provided components are limited and do not correspond to OpenLayers. Our library aims to display Google maps and OpenLayers without any loss of components. Therefore, the library implements new set of features and components for Google maps. Google Maps API native components encapsulate access to data fields. In addition, once created the features on the map are hard to select and change. The users familiar to OpenLayers expect to receive the same functionality and tools on any map. Consequently, a new class hierarchy is developed on top of Google Maps' base classes for compatibility with OpenLayers features. During the development the OpenLayers' class hierarchy model had been followed in order to achieve smoother integration. That way the developers receive the same behaviour on a map and continue to work with familiar class hierarchy. Meanwhile, OpenLayers and Google maps are displayed via native library code, thereby they are very easy to support and handle.

2. Functional features

Our library consists of two main components: Google map API extender and abstract map manager.

2.1. Google map API extender

Google Maps API exposes an easy to extend class hierarchy [1]. Google map API extender is a Javascript library developed on the top of Google Maps API base classes via Javascript prototyping mechanism [7]. The Google map API extender layer follows OpenLayers API's class hierarchy model. Thereby, the already implemented functionality in business applications has been kept with minimal possibility of errors. In this way, the library can be integrated into already working solutions effortlessly.

- Map class
- Layer class
- Features

2.1.1. Map class

The google.maps.Map class is extended within Javascript prototype chaining mechanism, that way the used Google map object supports native OpenLayers maps functionality.

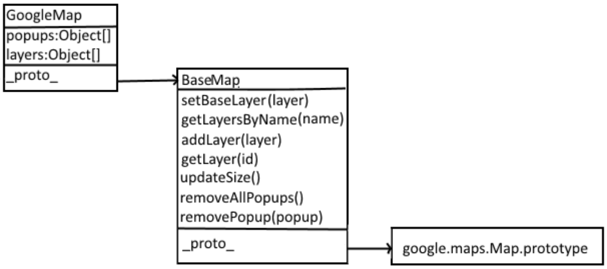


Figure 1. Maps class diagram

An abstract BaseMap class is designed. It inherits from google.maps.Map.prototype. The BaseMap class receives from its base class a zoom functionality, an ability to set a center and manipulate a map. We included additional methods within the abstract class' scope. These methods follow Openlayers.Map's most popular functionality related to layers management. The new methods for handling layers are: "setBaseLayer", "getLayersByName", "addLayer", "getLayer", "updateSize". These methods allow developers to manage different type of data on a Google map: Street view, Hybrid view, Satellite view, Traffic layer and custom features. The abstract class is used as a prototype of GoogleMap class, which adds instance's specific fields. The GoogleMap class itself is used to instantiate Google map in a specified container.

2.1.2. Layer class

The OpenLayers API has two very important concepts: map and layer. The map is an object that stores information about projections, zoom level, units and other map settings. In the same time the data on a map is displayed via layer objects. The layer has information how to request and display data from any data source. As mentioned earlier, OpenLayers provides wrappers for Google Maps, Bing, OpenStreetMap and others. These wrappers are available through layer objects. In OpenLayers practice it is common to separate the data from different data sources in different layers. The described map manager is designed to work with already existing web business solutions, which makes the layers usage necessity. Working with layers has its advantages. It makes it safer to maintain set of features from a map without affecting the rest of the features. The layers support event handling and visibility control. The Google Maps API does not provide a layers management. It is up to developers to maintain layers over Google map. The exposed Google Maps class library allows handling of different types of data over the map within google.maps.Data class. Our maps module handles each google.maps.Data instance as a layer with a different type of components. We developed a specific class called GoogleLayer. It inherits from google.maps.Data. The map objects can contain a collection of different layers ("Street view", "Satellite view", "Hybrid view" and layers with components).

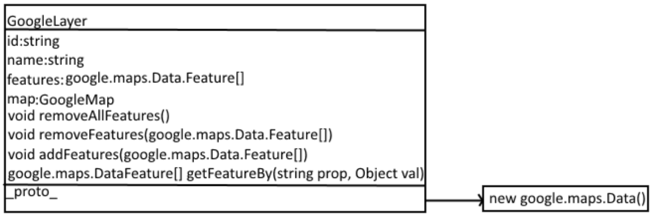

GoogleLayer
id:string
name:string
features: google.maps.Data.Feature[]
map:GoogleMap
void removeAllFeatures()
void removeFeatures(google.maps.Data.Feature[])
void addFeatures(google.maps.Data.Feature[])
google.maps.DataFeature[] getFeatureBy(string prop, Object val)
proto
new google.maps.Data()

Figure 2. Google Layer class diagram

The advantage to use google.maps.Data as a base layer class is that it has built-in functionality for features management. The derived functionality includes management and event handling of google.maps.Feature collections. The GoogleLayer implementation extends google.maps.Data class with methods that match OpenLayers.Layer.Vector definition. Within the extension methods the API has a direct access to the layer's features (Google Maps API hides the feature collection and provides access via iterator). At the end, the client code should not know what type of layer it is dealing with. It uses the same methods and receives the expected behaviour even on different maps.

2.1.3. Features

As mentioned above OpenLayers provides a wide variety of features on maps that includes geometry, drawing, vector, markers and others. The geometry features that handles POI (Point of interest) creation on a map are among the most important ones. Google Maps API provides two types of geometry representation over the map: within google.maps namespace or within google.maps.Data namespace. Here it is important to note that google.maps.Data layer contains of google.maps.Data features. On the other hand, geometry objects exposed via google.maps namespace have more flexible behaviour. google.maps.Polygon, Polyline, Circle, etc. provide the ability to render dynamically objects on a map, to turn on object editing mode and to handle long list of events. The implemented features classes use the advantages of google.maps geometry objects on a google.maps.Data layer. A separate class is created for each component. The new classes are extending google.map.Data.Feature in order to respect Google Maps interface requirements. To achieve rendering of dynamic objects on a google.maps.Data layer we designed the derived classes with google.maps geometry field.

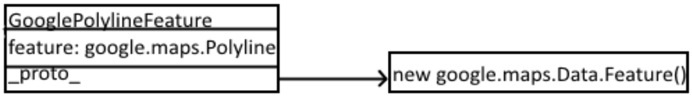


Figure 3. Feature common class diagram

Our Google map API extender exposes classes for Polygon, Polyline, Circle, Marker, Drawing and Vector feature (additional fields are added when necessary) through the use of that class schema.

OpenLayers API provides OpenLayers.Feature.Vector class which instantiates custom objects on map. Within OpenLayers.Feature.Vector class the map can contain features in different style, source, etc. In addition, the OpenLayers.Popup class provides the ability to add custom html content to features on a map. The usage of these two classes makes it possible to maintain any kind of custom data on map.

Our map module creates a separate feature class that provides the same functionality as OpenLayers.Feature.Vector and OpenLayers.Popup called MapFeature. The MapFeature class extends google.maps.OverlayView. It is used as a prototype class for the Vector features on a map. google.maps.OverlayView is a special class which aims to display a custom overlay types over the Google map. MapFeature class implements "onAdd", "draw" and "onRemove" functions of google.maps.OverlayView. Within "draw" function the MapFeature class handles creation of DOM elements over the map and proper event attachment to those elements. "onRemove" manages detachment of DOM listeners and object destruction. The MapFeature class makes it possible to draw custom images, dynamic html content, different styles of containers and others over the map with easy event handling and selection.

2.2. Maps manager

After we have the functionality to create components with the same look and feel over OpenLayers and Google maps we need a map manager to control the lifecycle of used map. The module provides the ability to switch easily between different maps and used API. This functionality is implemented in a bridge layer which verifies that only one map is used at each moment, the layer disposes unused objects, components and event listeners. It is responsible for saving and restoring the map state. This includes memorization of drawn popups, geometry features, event listeners, etc. The layer is responsible for the recreation of already drawn components on a map. The module aims to deliver similar look and feel on different maps.

- API management

- Maps management
- States management
- Positions

2.2.1. API management

A new class called MapFactory is created to give access to implemented functionality. Within Javascript namespace encapsulation the MapFactory instances hide used map API (Google or OpenLayers). The namespace exposes public interface that uses current map API.

2.2.2. Maps management

When a map object is requested from client code the manager layer returns a GoogleMap or OpenLayers.Map object (GoogleMap and OpenLayers.Map classes are compatible). During module implementation we faced the problem with destruction of google.maps.Map object. Google Maps API does not provide the functionality to manually destroy a map, map release is handled via garbage collector on page leave. This is a problem for solutions that provides functionality to change dynamically current map. Our solution proposes a dynamic switch between OpenLayers map and Google map. The map manager prevents memory leaks: when GoogleMap should be hidden, all components, layers, etc. on a map and their event listeners are disposed. Google Maps API base classes, that are used in Google map API extender, provide an easy way to clear instance references. When all clean-ups are done, the map manager holds lightweight google.maps.Map proxy object. Thereby a fully functional map object can be loaded within the same reference on demand.

2.3. States management

The bridge layer is responsible for management of the map state. During the switch between OpenLayers and Google maps (and vice versa) it saves center, zoom, event listeners and other map settings. All of the listed settings should be restored when the switching is completed. The end user should not be aware of a different map object usage.

2.3. Positions

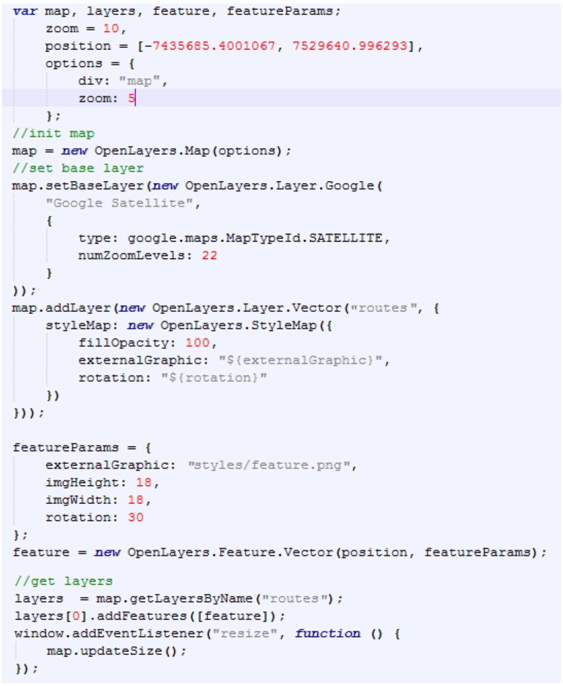
The OpenLayers native maps have different projection then Google map. The usage of completely independent map objects avoids the projection gap between coordinates over different maps. During map state recreation projection helper function is used to indicate whether it is necessary to transform given position. The functionality returns regular map position as a result.

3. Usage

In the following section the usage of described map module will be demonstrate.

3.1. Create maps within OpenLayers API only

Figure 4 demonstrates map initialization with OpenLayers API. The code instantiates map with base Google Satelitte layer. In addition, a vector with custom image background is displayed on the map.

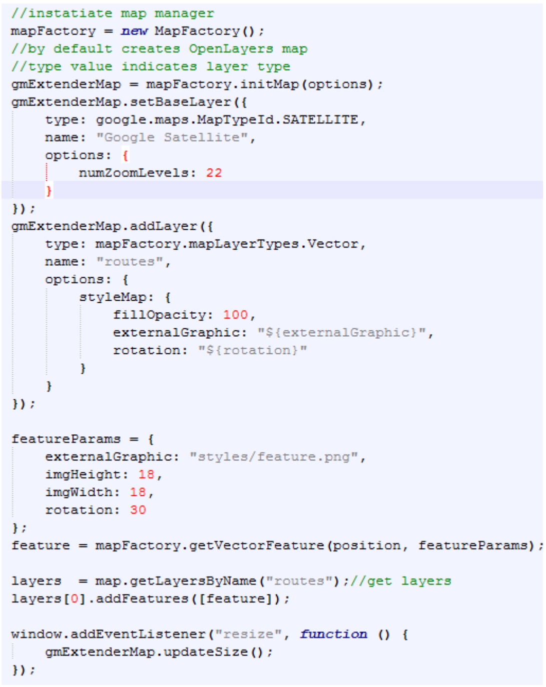


```
var map, layers, feature, featureParams;
    zoom = 10,
    position = [-7435685.4001067, 7529640.996293],
    options = {
        div: "map",
        zoom: 5
    };
//init map
map = new OpenLayers.Map(options);
//set base layer
map.setBaseLayer(new OpenLayers.Layer.Google(
    "Google Satellite",
    {
        type: google.maps.MapTypeId.SATELLITE,
        numZoomLevels: 22
    }
));
map.addLayer(new OpenLayers.Layer.Vector("routes", {
    styleMap: new OpenLayers.StyleMap({
        fillOpacity: 100,
        externalGraphic: "${externalGraphic}",
        rotation: "${rotation}"
    })
}));

featureParams = {
    externalGraphic: "styles/feature.png",
    imgHeight: 18,
    imgWidth: 18,
    rotation: 30
};
feature = new OpenLayers.Feature.Vector(position, featureParams);

//get layers
layers  = map.getLayersByName("routes");
layers[0].addFeatures([feature]);
window.addEventListener("resize", function () {
    map.updateSize();
});
```

Figure 4. OpenLayers map initialization code snippet

3.2. Create maps within MapFactory maps manager

Figure 5 demonstrates the usage of our library for dynamic maps. Created map follows OpenLayers basic concepts for map creation and layer handling. The code instantiates map with base Google Satellite layer. Custom vector feature is created on a map. The developers do not need to know the actual map API. They continue to work with familiar class hierarchy and components. The MapFactory instance is responsible for displaying data on a map and interactions with Google Maps and OpenLayers API.



```
//instatiate map manager
mapFactory = new MapFactory();
//by default creates OpenLayers map
//type value indicates layer type
gmExtenderMap = mapFactory.initMap(options);
gmExtenderMap.setBaseLayer({
    type: google.maps.MapTypeId.SATELLITE,
    name: "Google Satellite",
    options: {
        numZoomLevels: 22
    }
});
gmExtenderMap.addLayer({
    type: mapFactory.mapLayerTypes.Vector,
    name: "routes",
    options: {
        styleMap: {
            fillOpacity: 100,
            externalGraphic: "${externalGraphic}",
            rotation: "${rotation}"
        }
    }
});

featureParams = {
    externalGraphic: "styles/feature.png",
    imgHeight: 18,
    imgWidth: 18,
    rotation: 30
};
feature = mapFactory.getVectorFeature(position, featureParams);

layers  = map.getLayersByName("routes");//get layers
layers[0].addFeatures([feature]);

window.addEventListener("resize", function () {
    gmExtenderMap.updateSize();
});
```

Figure 5. MapFactory map initialization code snippet

4. Conclusion

The described functionality solves future issues related to library updates that cannot be foreseen. It provides the ability to have a rich variety of compatible components over OpenLayers and Google map. The module uses native API methods to show native components. The main goal – to keep the usage of Google maps and OpenLayers rich functionality – is accomplished. The underling implementation remains hidden for the developers and the integration of our map module needs minimum effort and provides safer way to maintain maps.

Acknowledgements

This work is partially supported by the FP17-FMI-008 project of the Scientific Fund of the University of Plovdiv “Paisii Hilendarski”, Bulgaria.

References

[1] Google Maps Javascript API V3 Reference, **https://developers.google.com/maps/documentation/javascript/reference**, retrieved on January 2016

[2] OpenLayers: Free Maps for the Web: **http://openlayers.org/two/**

[3] **https://github.com/openlayers**, retrieved on July 2017

[4] Mapping usage statistics, **http://trends.builtwith.com/mapping,** retrieved on August 2017

[5] OpenLayers 3 on Git, **https://github.com/openlayers/ol3/issues/4443**, retrived on July 2017

[6] Reference to community discussion for problems that occurred with Google Maps layer visualization on OpenLayers 2, **https://github.com/openlayers/ol2/issues/1450#issuecomment-157611045,** retrieved on December 2015

[7] Inheritance and prototype chain, **https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain**, retrieved on December 2015

Faculty of Mathematics and Informatics
Plovdiv University
236 Bulgaria Blvd, Plovdiv 4003, Bulgaria
E-mail: teodoragardjeva@gmail.com, assen@uni-plovdiv.bg,
nikolayp@uni-plovdiv.bg, angelg@uni-plovdiv.bg

JAVASCRIPT БИБЛИОТЕКА ЗА ВИЗУАЛИЗАЦИЯ НА ГЕОГРАФСКИ КАРТИ

Теодора Гарджева, Асен Рахнев, Николай Павлов, Ангел Голев

Резюме. Представяме JavaScript библиотека за визуализация на географски карти в уеб базирани приложения, синтактично и функционално съвместима с OpenLayers 2. Библиотеката преодолява трудностите, породени от непрекъснатото развитие на популярни инструменти като Google Maps API и Bing Maps API. Поддържат се карти както на OpenLayers, така и на Google. Описан е обектно-ориентираният модел на библиотеката и нейната функционалност. Илюстрирано е как библиотеката може да бъде използвана в крайни приложения. Библиотеката е внедрена в реален проект за управление на автопарк.