
Wicket

build unknown cdnjs v1.3.8

Wicket is a lightweight library for translating between Well-Known Text (WKT) and various client-side mapping frameworks: * Leaflet (demo) * Google Maps API (demo) * ESRI ArcGIS JavaScript API * Potentially any other web mapping framework through serialization and de-serialization of GeoJSON (with `JSON.parse`)

The core Wicket library and the Leaflet extension are both compatible with Node.js; the Google Maps and ArcGIS API extensions will not work in Node.js because they require a browser.

If you are looking for Apache Wicket, the web-app development framework for Java, you'll find it [here](#).

License

Wicket is released under the GNU General Public License version 3 (GPLv3). Accordingly:

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Installation

```
1 $ npm install wicket
```

Example

The following examples work in any of the mapping environments, as Wicket has a uniform API regardless of the client-side mapping library you're using.

```
1 // Create a new Wicket instance
2 var wkt = new Wkt.Wkt();
3
4 // Read in any kind of WKT string
5 wkt.read("POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))");
6
7 // Or a GeoJSON string
8 wkt.read('{"coordinates": [[[30, 10], [10, 20], [20, 40], [40, 40],
9           [30, 10]]], "type": "Polygon"}');
```

```
10 // Access and modify the underlying geometry
11 console.log(wkt.components);
12 // "[ [ {x: 30, y: 10}, {x: 10, y: 30}, ...] ]"
13 wkt.components[0][1].x = 15;
14
15 wkt.merge(new Wkt.Wkt('POLYGON((35 15,15 25,25 45,45 45,35 15))'));
16 wkt.write();
17 // MULTIPOLYGON(((30 10,10 20,20 40,40 40,30 10)),((35 15,15 25,25
    45,45 45,35 15)))
18
19 // Create a geometry object, ready to be mapped!
20 wkt.toObject();
21
22 // Convert to GeoJSON
23 wkt.toJson(); // Outputs an object
24 JSON.stringify(wkt.toJson()); // Outputs a string
```

Wicket will read from the geometry objects of any mapping client it understands. **Note:** Don't use the `deconstruct()` method! This is used internally by `Wkt.Wkt()` instances. Use `fromObject()` instead, as in the following example.

```
1 var wkt = new Wkt.Wkt();
2
3 // Deconstruct an existing point feature e.g. google.maps.Marker
  instance
4 wkt.fromObject(somePointObject);
5
6 console.log(wkt.components);
7 // "[ {x: 10, y: 30} ]"
8
9 // Serialize a WKT string from that geometry
10 wkt.write();
11 // "POINT(10 30)"
```

See Also

- [wellknown](#)
- [OpenLayers WKT](#)
- [wkt-parser](#)

Dependencies and Build Information

Wicket has zero dependencies, however, JSON parsing (from strings) is not provided. Wicket looks for the function `JSON.parse`, which is provided in most modern browsers (get it with this library, if you need to support older browsers).

Minified versions can be generated via:

```
1 npm run build
```

Testing

```
1 npm test
```

Documentation

Read the documentation [here](#). Documentation can be generated with JSDoc 3.

```
1 git clone git://github.com/jsdoc3/jsdoc.git
2 ./jsdoc /var/www/static/wicket/wicket.src.js
```

Or, with Node installed:

```
1 sudo npm install -g git://github.com/jsdoc3/jsdoc.git
2 jsdoc /var/www/static/wicket/wicket.src.js
```

Either way, make sure you invoke `jsdoc` from a directory in which you have write access; it will output documentation to your current working directory.

Colophon

Motivation

Wicket was created out of the need for a lightweight Javascript library that can translate Well-Known Text (WKT) strings into geographic features. This problem arose in the context of OpenClimateGIS, a web framework for accessing and subsetting online climate data.

OpenClimateGIS emits WKT representations of user-defined geometry. The API Explorer allowed users to define arbitrary areas-of-interest (AOIs) and view predefined AOIs on a Google Maps API instance. So, initially, the problem was converting between WKT strings and Google Maps API features. While other mapping libraries, such as OpenLayers, have very nice WKT libraries built-in, the Google Maps API, as of this writing, does not. In the (apparent) absence of a lightweight, easy-to-use WKT library in Javascript, I set out to create one.

That is what Wicket aspires to be: lightweight, framework-agnostic, and useful. I hope it achieves these goals. If you find it isn't living up to that and you have ideas on how to improve it, please fork the code or drop me a line.

Acknowledgements

Wicket borrows heavily from the experiences of others who came before us:

- The OpenLayers 2.7 WKT module (OpenLayers.Format.WKT)
- Chris Pietshmann’s article on converting Bing Maps shapes (VEShape) to WKT
- Charles R. Schmidt’s and the Python Spatial Analysis Laboratory’s (PySAL) WKT writer

Conventions

The base library, wicket.js, contains the Wkt.Wkt base object. This object doesn’t do anything on its own except read in WKT strings, allow the underlying geometry to be manipulated programmatically, and write WKT strings. By loading additional libraries, such as wicket-gmap3.js, users can transform between between WKT and the features of a given framework (e.g. google.maps.Polygon instances). The intent is to add support for new frameworks as additional Javascript files that alter the Wkt.Wkt prototype.

To extend Wicket, nominally by writing bindings for a new mapping library, add a new file with a name like wicket-libname.src.js (and corresponding minified version wicket-libname.js) where “libname” is some reasonably short, well-known name for the mapping library.

Concepts

WKT geometries are stored internally using the following convention. The atomic unit of geometry is the coordinate pair (e.g. latitude and longitude) which is represented by an Object with x and y properties. An Array with a single coordinate pair represents a single point (i.e. POINT feature):

```
1 [ {x: -83.123, y: 42.123} ]
2
3 // POINT(-83.123 42.123)
```

An Array of multiple points (an Array of Arrays) specifies a “collection” of points (i.e. a MULTIPOINT feature):

```
1 [
2   [ {x: -83.123, y: 42.123} ],
3   [ {x: -83.234, y: 42.234} ]
4 ]
5 // MULTIPOINT(-83.123 42.123,-83.234 42.234)
```

An Array of multiple coordinates specifies a collection of connected points in an ordered sequence (i.e. LINESTRING feature):

```
1  [  
2      {x: -83.12, y: 42.12},  
3      {x: -83.23, y: 42.23},  
4      {x: -83.34, y: 42.34}  
5  ]  
6  // LINESTRING(-83.12 42.12,-83.23 42.23,-83.34 42.34)
```

An Array can also contain other Arrays. In these cases, the contained Array(s) can each represent one of two geometry types. The contained Array might represent a single polygon (i.e. POLYGON feature):

```
1  [  
2      [  
3          {x: -83, y: 42},  
4          {x: -83, y: 43},  
5          {x: -82, y: 43},  
6          {x: -82, y: 42},  
7          {x: -83, y: 42}  
8      ]  
9  ]  
10 // POLYGON(-83 42,-83 43,-82 43,-82 42,-83 42)
```

The above example cannot represent a LINESTRING feature (one of the few type-based constraints on the internal representations), however it may represent a MULTILINESTRING feature. Both POLYGON and MULTILINESTRING features are internally represented the same way. The difference between the two is specified elsewhere (in the Wkt instance's type) and must be retained. In this particular example (above), we can see that the first coordinate in the Array is repeated at the end, meaning that the geometry is closed. We can therefore infer it represents a POLYGON and not a MULTILINESTRING even before we plot it. Wicket retains the *type* of the feature and will always remember which it is.

Similarly, multiple nested Arrays might represent a MULTIPOLYGON feature:

```
1  [  
2      [  
3          [  
4              {x: -83, y: 42},  
5              {x: -83, y: 43},  
6              {x: -82, y: 43},  
7              {x: -82, y: 42},  
8              {x: -83, y: 42}  
9          ]  
10     ],  
11     [  
12         [  
13             {x: -70, y: 40},  
14             {x: -70, y: 41},  
15             {x: -69, y: 41},  
16             {x: -69, y: 40},  
17             {x: -70, y: 40}
```

```

18     ]
19   ]
20 ]
21 // MULTIPOLYGON((( -83 42,-83 43,-82 43,-82 42,-83 42),(-70 40,-70
    41,-69 41,-69 40,-70 40)))

```

Or a POLYGON with inner rings (holes) in it where the outer ring is the polygon envelope and comes first; subsequent Arrays are inner rings (holes):

```

1  [
2    [
3      {x: 35, y: 10},
4      {x: 10, y: 20},
5      {x: 15, y: 40},
6      {x: 45, y: 45},
7      {x: 35, y: 10}
8    ],
9    [
10     {x: 20, y: 30},
11     {x: 35, y: 35},
12     {x: 30, y: 20},
13     {x: 20, y: 30}
14   ]
15 ]
16 // POLYGON((35 10,10 20,15 40,45 45,35 10),(20 30,35 35,30 20,20 30))

```

Or they might represent a MULTILINESTRING where each nested Array is a different LINESTRING in the collection. Again, Wicket remembers the correct *type* of feature even though the internal representation is ambiguous.