

Geomajas REST face guide

Geomajas Developers and Geosparc

Geomajas REST face guide

by Geomajas Developers and Geosparc

1.16.2

Copyright © 2010-2014 Geosparc nv

Table of Contents

1. Introduction	1
2. Configuration	2
1. Dependencies	2
2. Web MVC configuration	2
3. Usage	3
1. Reading a single feature	3
2. Reading multiple features	3
3. OpenLayers configuration	5

List of Tables

3.1. HTTP parameters	3
3.2. HTTP parameters	4

List of Examples

2.1. Plug-in dependency	2
-------------------------------	---

Chapter 1. Introduction

The REST face provides access to the Geomajas server through a RESTful protocol. The current implementation provides read-only access to the various vector layers configured on the server. Our protocol implementation was largely inspired by the Mapfish protocol [<http://trac.mapfish.org/trac/mapfish/wiki/MapFishProtocol>], which should be in line with - or at least supported by - the OpenLayers client implementation. We also support clients of FeatureServer [<http://featureserver.org/>], which has a much more limited parameterset. As far as we know, there is currently no REST standard for feature services, although there is some consensus on the rest principles (using uri extensions to determine the output format, exposing single features as uris, etc...).

Chapter 2. Configuration

The configuration of the REST plugin consists of a maven part and a web MVC part.

1. Dependencies

Make sure you include the plug-in in your project. If you are using Maven, add the following dependency to your pom:

Example 2.1. Plug-in dependency

```
<dependency>
  <groupId>org.geomajas</groupId>
  <artifactId>geomajas-face-rest</artifactId>
  <version>1.16.2</version>
</dependency>
```

2. Web MVC configuration

The web configuration is by default based on the bean name URL handler mapping. This means that the name of the scanned controller beans determines the URL base that points to the REST services. More specifically, the `RestController` class (see below) has an annotation that tells the `DispatcherServlet` to map all urls that start with `http://{host}:{port}/{context}/{dispatcher}/rest/` to the REST service, where `host`, `port` and `context` should be replaced with the host, port and web context of your deployment and `{dispatcher}/*` is the url pattern of the `DispatcherServlet` mapping (as configured in `web.xml`).

Warning

the current faces depend on the mapping of `/d/*` to the `DispatcherServlet` !

```
@Controller("/rest/**")
public class RestController {
    @RequestMapping(value = "/rest/{layerId}/{featureId}.json", method = RequestM...
    ...
}
```

Further narrowing of the url mapping to class methods is done through the `RequestMapping` annotations. For a thorough explanation of the various url parameters we refer to the next chapter, which explains the usage of the REST face as a client. Unless you absolutely want to change the url path, there is no need to override the default configuration. If you do want to change this, however, one possibility is to add additional mappings by configuring a `SimpleUrlHandlerMapping` bean, which allows mapping of arbitrary paths to controller beans.

Chapter 3. Usage

The REST plugin exposes a RESTful web interface towards compatible clients. In this chapter we will give an overview of all available HTTP operations and their respective parameters. The base uri for the REST service is `http://{host}:{port}/{context}/{dispatcher}` where `{dispatcher}/*` is the url mapping for the `DispatcherServlet` (as configured in `web.xml`). We will refer to this as `http://base`.

Warning

the current faces depend on the mapping of `/d/*` to the `DispatcherServlet` !

Although extensible to multiple response formats, the REST service is currently limited to GeoJSON only. The GeoJSON specification can be found at <http://geojson.org/geojson-spec.html> [???].

1. Reading a single feature

The method and request uri for reading a single feature is:

```
GET http://base/rest/{layerId}/{featureId}.{format}
```

The url parameters are:

- `layerId` : the server layer id of the vector layer
- `featureId` : the id of the feature
- `format` : the format of the returned feature, possible values are: json, kml, shp, txt

If successful, the response is as follows:

- 200 OK
- a single feature in GeoJSON, KML, shapefile or text format

If the feature does not exist, the response is as follows:

- 404 Not Found

If something else went wrong, the response is:

- 500 Internal server error
- the error message of the `RestException`

The following parameters are supported:

Table 3.1. HTTP parameters

Name	Value	Description
<code>no_geom</code>	<code>true false</code>	if true, the returned feature has no geometry (<code>"geometry": null</code>)
<code>attrs</code>	<code>{field1}[,{field2},...]</code>	a list of attributes to restrict the list of properties returned in the feature

2. Reading multiple features

The method and request uri for reading multiple features is:

GET `http://base/rest/{layerId}`

The url parameters are:

- `layerId` : the server layer id of the vector layer

If successful, the response is as follows:

- 200 OK
- a GeoJSON feature collection with 0 or more features

If the feature does not exist, the response is as follows:

- 404 Not Found

If something else went wrong, the response is:

- 500 Internal server error
- the error message of the `RestException`

The following parameters are supported:

Table 3.2. HTTP parameters

Name	Value	Description
<code>no_geom</code>	<code>true false</code>	if true, the returned feature has no geometry ("geometry": null)
<code>epsg</code>	<code>{num}</code>	the EPSG code of the box value and the returned geometries
<code>format</code>	<code>json shp txt kml</code>	The format of the response, default is json
<code>attrs</code>	<code>{field1}[, {field2}, ...]</code>	a list of attributes to restrict the list of properties returned in the feature
<code>box bbox</code>	<code>{xmin,ymin,xmax,ymax}</code>	a list of coordinates representing a bounding box, the coords' projection system can be specified with the epsg parameter (bbox is an alias to box)
<code>maxFeatures limit</code>	<code>{num}</code>	limit the number of features to num features (maxfeatures is an alias to limit)
<code>offset</code>	<code>{num}</code>	skip num features
<code>order_by</code>	<code>{field}</code>	order the features using field
<code>dir</code>	<code>DESC ASC</code>	determine the ordering direction (applies only if order_by is specified)
<code>queryable</code>	<code>{field1}[, {field2}, ...]</code>	the names of the feature fields that can be used in the field query parameters (see next row)
<code>{field}__{query_op}</code>	<code>{value}</code>	specify a filter expression, field must be in the list of

Name	Value	Description
		fields specified by queryable, supported query_op's are: <ul style="list-style-type: none">• eq: equal to• ne: not equal to• lt: lower than• lte: lower than or equal to• gt: greater than• gte: greater than or equal to• like: sql like for string values

The parameters lat, lon, tolerance have not yet been implemented.

3. OpenLayers configuration

OpenLayers [<http://openlayers.org/>] is the most popular web mapping javascript library and supports the REST protocol for feature access. To connect with an OpenLayers client, add the following layer to your map:

```
var layer = new OpenLayers.Layer.Vector("GeoJSON", {
  strategies: [new OpenLayers.Strategy.Fixed()],
  protocol: new OpenLayers.Protocol.HTTP({
    url: "http://base/rest/{layerId}", // replace with the actual url for your
    format: new OpenLayers.Format.GeoJSON()
  })
});
```

An example web application with OpenLayers can be found in the [geomajas-face-rest-example](#) project.