Geomajas Server geocoder plugin guide

Geomajas Developers and Geosparc

Geomajas Server geocoder plugin guide

by Geomajas Developers and Geosparc

1.18.1

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Introduction

The geocoder plugin provides a standard mechanism to convert text, representing an address or point of interest, in a map location.

For this purpose a command "command.geocoder.GetLocationForString" is provided which allows you to do the conversion

This input string is converted to a bounding box which is intended to be zoomed to. A marker may be displayed at the center of that area.

The actual conversion is done using a pluggable list of geocoder service. Some standard implementations are provided. Either the first services which produces a match wins, or the results may be combined.

1. How does it work?

The GetLocationForStringCommand class handles a request to get a location for a string.

Figure 1.1. GetLocationForStringCommand overview

This string is first cut into relevant parts, for example "London, UK" may be split into parts "London" and "UK". This is handled by the SplitGeocoderStringService. The default implementation uses comma as separator and removes whitespace between the parts.

After that, each of the configured GeocoderServices is given a chance to convert the strings into a list of locations. If the geocoder service returns one location, it is considered *matched*. When it returns multiple locations, the search term is considered ambiguous and the locations are considered as alternatives. You can configure whether all geocoders need to be given a chance to find the location or whether it should stop as soon as one service has returned at least one location.

The geocoder services can return either an area (bounding box) or a point. When they returned a point, this is extended to an area centered around that point.

At the end, the result is prepared for return. The matching locations are combined using the configured CombineResultService. The default implementation uses the union of the area. When there were no matches, then all alternatives are returned.

The location also contains the canonical search string.

Chapter 2. Use

How to use the geocoder plugin.

1. Command

The main access point for the functionality which is provided by this plugin is the GetLocationForString command.

Table 2.1. GetLocationForStringCommand

GetLocationForStringCommand	
Registry key	command.geocoder.GetLocationForString
Module which provides this command	geomajas-plugin-geocoder
Request object class	org.geomajas.plugin.geocoder.command.dto.GetLocationForStringRequest
Parameters	• *location*: string which a geometric location should be searched for. • *crs*: the coordinate reference system which should be used for the response. • *servicePattern*: regular expression which allows you to select which geocoder service are used to search for the result. By default all configured services are used. • *locale*: locale used for the search if known. • *maxAlternatives*: maximum number of alternatives to return in the reply. This defaults to 50.
Description	This command allows you to find a map location from a string representation.
Response object class	org.geomajas.plugin.geocoder.command.dto.GetLocationForStringResponse
Response values	• *locationFound*: indicates whether there was a match. • *canonicalLocation*: preferred string for searching the matched location. Can be the same as the location request parameter. • *center*: center of the matched area. • *bbox*: matched area, the area which the map should probably zoom to. • *geocoderName*: name of the geocoder which produces the result. Only available when there was only one matched result. • *userData*: any additional object which was included in the match result. Only available when there was only one matched result. • *alternatives*: when no match was found, this field may contain a list of locations which may match the request. Each alternative contains the fields above.

As part of other plugins, tests or code which has the back-end in the same VM, this can be run as in listing Example 2.1, “Usage of the geocoder command”.

Example 2.1. Usage of the geocoder command

```
@Autowired
private CommandDispatcher commandDispatcher;

@Test
public void oneResultTest() throws Exception {
    GetLocationForStringRequest request = new GetLocationForStringRequest()
    request.setCrs("EPSG:4326");
    request.setLocation("booischot");

    CommandResponse commandResponse = commandDispatcher.execute(GetLocationF
            "en");
}
```

For details on calling this command from inside a client, see the specific client's documentation.

Chapter 3. Configuration

Configuration for the geocoder plugin.

1. Dependencies

Make sure sure you include the correct version of the plugin in your project. Use the following excerpt (with the correct version) in the dependencyManagement section of your project:

```
 <dependency>
    <groupId>org.geomajas.project</groupId>
    <artifactId>geomajas-project-server</artifactId>
    <version>1.18.1</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

If you are using geomajas-dep, this includes the latest released version of the caching plugin (at the time of publishing of that version). If you want to overwrite the caching plugin version, make sure to include this excerpt *before* the geomajas-dep dependency.

You can now include the actual dependency without explicit version.

Example 3.1. Plugin dependency

```
<dependency>
    <groupId>org.geomajas.plugin</groupId>
    <artifactId>geomajas-plugin-geocoder</artifactId>
</dependency>
```

2. Static regex geocoder service configuration

The StaticRegexGeocoderService allows you to define the combinations of string to match and the locations directly in the configuration file.

The strings to match are specified using regular expressions[1]to allow more flexibility. Listing Example 3.2, “Base configuration for StaticRegexGeocoderService” shows a base configuration. You have to use the geocoderInfo property to configure the geolocator. This is done using a StaticRegexGeocoderInfo object which contains the coordinate space name (EPSG:900913 in this case, which is Mercator) and define the location mappings.

Example 3.2. Base configuration for StaticRegexGeocoderService

```
<bean name="staticRegexGeocoderService" class="org.geomajas.plugin.geocoder.serv
    <property name="geocoderInfo">
        <bean class="org.geomajas.plugin.geocoder.api.StaticRegexGeocoderInfo">
            <property name="crs" value="EPSG:900913"/>
            <property name="locations">
                <list>
                    <ref bean="BooischotShort"/>
                </list>
            </property>
```

[1]see http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html

```
        </bean>
    </property>
</bean>
```

The location mappings themselves are contained in StaticRegexGeocoderLocationInfo instances. You have to specify the strings toMatch, and a location as either point of bounding box. You can specify the canonical form for the search.

In listing Example 3.3, “Defining a point” you see a definition which will match a single location string starting with second. As this is done case independently, some examples of matching strings are "second" and SECondary". It indicates a point with coordinates (10000,10000).

Example 3.3. Defining a point

```
<bean class="org.geomajas.plugin.geocoder.api.StaticRegexGeocoderLocationInfo">
    <property name="toMatch">
        <list>
            <value>second.*</value>
        </list>
    </property>
    <property name="canonical">
        <list>
            <value>secondService</value>
        </list>
    </property>
    <property name="x" value="10000"/>
    <property name="y" value="10000"/>
</bean>
```

The location info object can also be used to match an area. In listing Example 3.4, “Defining an area bbox” you see the location bounding box defined using the bbox property. If you would accidentally define both a bounding box and point coordinates, then the bounding box will be used for the result.

Example 3.4. Defining an area bbox

```
<bean class="org.geomajas.plugin.geocoder.api.StaticRegexGeocoderLocationInfo">
    <property name="toMatch">
        <list>
            <value>bbox</value>
        </list>
    </property>
    <property name="bbox">
        <bean class="org.geomajas.geometry.Bbox">
            <property name="x" value="0"/>
            <property name="y" value="50000"/>
            <property name="width" value="100000"/>
            <property name="height" value="80000"/>
        </bean>
    </property>
</bean>
```

A location can also include extra data in the result. You need to wrap this data in a subclass of ClientUserDataInfo. The object to be returned can be defined using the userData property.

Example 3.5. Defining an area and user data

```
<bean class="org.geomajas.plugin.geocoder.api.StaticRegexGeocoderLocationInfo">
    <property name="toMatch">
        <list>
```

```
            <value>bla</value>
        </list>
    </property>
    <property name="bbox">
        <bean class="org.geomajas.geometry.Bbox">
            <property name="x" value="30000"/>
            <property name="y" value="50000"/>
            <property name="width" value="10000"/>
            <property name="height" value="10000"/>
        </bean>
    </property>
    <property name="userData">
        <bean class="org.geomajas.plugin.geocoder.service.UserDataTestInfo">
            <property name="value" value="xobb"/>
        </bean>
    </property>
</bean>
```

The toMatch property contains a list of strings which need to be matched in order. The matching checks every string in the location strings for a matching string in the toMatch list, in order. The matching is case independent and always matches the entire string. A level can be marked as optional in the location strings by using a question marks as prefix for the regular expression. The question mark is removed before the actual evaluation of the regular expression.

As an example we will apply the example in listing Example 3.6, “Multiple strings to match.” to a couple of data sets.

- ["Belgium", "Antwerpen", "Booischot"]: matches, all three parts match the specific regular expressions.
- ["Booischot", "Antwerpen", "Belgium"]: no match as the "BE.*" regular expression does not match "Booischot".
- ["BE", "Booischot"]: matches, the "Antwerp.*" regular expression is marked as option using the "?" prefix.
- ["Belgium","Antwerpen"]: does not match as the "Booischot" regular expression is not matched for lack of input strings.
- ["Belgium", "Antwerpen", "Booischot", "Broekmansstraat"]: not matches as the last string "Broekmansstraat" does not have a matching regular expression.

Example 3.6. Multiple strings to match.

```
<bean name="BooischotStrict" class="org.geomajas.plugin.geocoder.api.StaticRege
    <property name="toMatch">
        <list>
            <value>Be.*</value>
            <value>?Antwerp.*</value>
            <value>Booischot</value>
        </list>
    </property>
</bean>
```

For this last case, where smaller divisions are not know (in this case the street name), you can end the list of regular expressions with "**" (see listing Example 3.7, “Multiple strings to match with open end.”). This will assure that any remaining strings from the input are discarded if any are remaining. This would assure that the last case in the previous list matches. The other cases would still have the same result.

Example 3.7. Multiple strings to match with open end.

```
<bean name="Booischot" class="org.geomajas.plugin.geocoder.api.StaticRegexGeoco
    <property name="toMatch">
        <list>
            <value>Be.*</value>
            <value>?Antwerp.*</value>
            <value>Booischot</value>
            <value>**</value>
        </list>
    </property>
</bean>
```

3. GeoNames service configuration

The GeonamesGeocoderService uses the search web service at geonames.org to handle the geocoder requests. You can only configure the "userName" property which is the geonames user which is registered to access the service. This can either be passed directly or using the "userNameProperty" which indicates the property which contains the user name to use. Defining the service is pretty straightforward.

Example 3.8. Defining the Geonames geocoder service

```
<bean class="org.geomajas.plugin.geocoder.service.GeonamesGeocoderService" >
    <property name="userName" value="geomajasHudson" />
</bean>
```

The GeoNames service never returns more than 50 results.

When the initial query returned no results, it will retry the search using fuzzy matching.

4. Yahoo! PlaceFinder geocoder service configuration

This uses the Yahoo! PlaceFinder service (http://developer.yahoo.com/geo/placefinder/). When using this geocoder, you need a appid from Yahoo! and you have to make sure you comply with their terms of use.

To use the geocoder, just create the bean and set the appId.

Example 3.9. Configuring the Yahoo! PlaceFinder geocoder

```
<bean name="ypf" class="org.geomajas.plugin.geocoder.service.YahooPlaceFinderGe
    <property name="appIdProperty" value="YahooAppId" />
</bean>
```

There are a couple of properties which influence how the appId can be passed:

- *appId*: you just define the appId in the configuration file.
- *appIdProperty*: the appId is read from the property which is specified. This can be helpful if you don't want to hardcode the property in your configuration files for some reason.
- *skipAppIdCheck*: normally an exception is thrown when the Yahoo! PlaceFinder geocoder is created without a appId. By setting this property to true, you can avoid this exception, making sure your application will run without the appId (though obviously no results can be found).

5. Type coordinate geocoder service configuration

This is a simplistic geocoder which allows the user to directly type the coordinate. To configure it, you just have to supply the default CRS used for the coordinates (if not specified, this defaults to EPSG:4326).

Example 3.10. TypeCoordinateService configuration

```
<bean name="tcs" class="org.geomajas.plugin.geocoder.service.TypeCoordinateServi
    <property name="defaultCrs" value="EPSG:900913" />
</bean>
```

The service accepts input strings like "4.77397 51.05125" to jump to a coordinate, using a space is used as separator between the ordinates. This uses the defaultCrs as configured. You can also explicitly specify the CRS by using a string like "4.77397 51.05125 crs:EPSG:4326".

Chapter 4. How-to

This chapter details the extension possibilities of the geocoder plugin.

1. How to allow users to switch between geocoder services

If you want your users to select between several geocoder services, you can use the servicePattern property in the command request to select the service.

You could for example configure both the Yahoo! PlaceFinder and GeoNames geocoder services. It is best that you provide explicit names to each service. You can now add a selection widget in your user interface. Depending on the selected value, you can use the setServicePattern() method of GeocoderWidget or alternatively on the GetLocationForStringRequest object of the command invocation to assure the selected geocoder service is used. Note that the service pattern is a regular expression. For alphanumerical names, just providing the name as pattern will work.

2. How to write a geocoder service

Writing a geocoder service is reasonable easy. All you have to do is create an implementation of the GeocoderService interface (listing Example 4.1, “Geocoder service interface definition”). The getCrs() method is used by Geomajas to know the coordinate system which is used for the results of your service. This is used to convert to the coordinate system of the client.

The name is used to select which services should be used for the search. It is recommended that you provide both a default name and a setter to allow users to change this.

Example 4.1. Geocoder service interface definition

```
public interface GeocoderService {

    /**
     * Name for the geocoder service. This name can be used to select which geo
     * search.
     * .
     * @return name for this geocoder
     */
    String getName();

    /**
     * CRS which is used for the results of this geocoder.
     *
     * @return CRS
     */
    CoordinateReferenceSystem getCrs();

    /**
     * Try to get a location for the strings passed. This can be either a coord
     * result object.
     *
     * @param location location strings, from general to more specific
     * @param maxAlternatives maximum number of alternatives which can be repli
     *        return more, but they will be discarded.
     * @param locale locale use for the location if known (can be null if not k
     * @return results objects, when only one this is a definite result, when s
```

```
     *         result are the alternatives. When no results an empty array or nu
     */
    GetLocationResult[] getLocation(List<String> location, int maxAlternatives,
}
```

The getLocation() method does the actual work of converting the location strings (ordered from most general, biggest area, to more specific) to a location. The method is expected to return null or an empty list if no results where found for the location string, or one object if the strings were matched, or multiple results when the matching was ambiguous and resulted in several alternatives.

The result itself contains the information from listing Example 4.2, “Fields which are defined in GetLocationResult”.

Example 4.2. Fields which are defined in GetLocationResult

```
public class GetLocationResult {

    private List<String> canonicalStrings;
    private Coordinate coordinate;
    private Envelope envelope;
    private String geocoderName;
    private ClientUserDataInfo userData;
```

The fields include:

- *canonicalStrings*: the preferred strings to use for the location. Can be null if no preferred string exists or it is not knows. In that case, the client will get the search strings as result.
- *coordinate*: the coordinate for the location. This field is only used when no envelope was given. The coordinate will be converted to the requested CRS and an area will be built around this point (according to command configuration).
- *envelope*: the bounding box for the location. This is the preferred result and has precedence over the coordinate field.
- *geocoderName*: name of te geocoder which produced the result. You don't have to set this, it will be set by the command.
- *userData*: any additional user data the geocoder may wish to return to the client.

3. How to write a service to split the input string

The geocoder command uses an instance of SplitGeocoderStringService to split and sort the initial search string. This should help to make the searching easier and assure the separator for location indicators can be configured and is geocoder service independent.

This way you can configure whether you prefer your users to type "London, UK", "UK, London", "London - UK" or something else.

You basically have to implement the service in listing Example 4.3, “Service interface for splitting the search string” and set that in the GeocoderInfo instance in your application context.

Example 4.3. Service interface for splitting the search string

```
public interface SplitGeocoderStringService {

    /**
```

```
     * Split the given string in a list of strings according to the separator c
     * biggest area should come first (assuming the original format had a notio
     *
     * @param location location to split
     * @return list of strings with split location
     */
    List<String> split(String location);

    /**
     * Combine the list of strings back into one string accoring to the conventi
     * the reverse of the split functions. As end result split(combine(split(x)
     *
     * @param matchedStrings strings to combine
     * @return combined string
     */
    String combine(List<String> matchedStrings);
}
```

You have to provide two methods, one for splitting and one for combining.

4. How to write a service to combine results

When multiple geocoder services found a match for the search string, an instance of CombineResultService is used to combine these results to one area. Two obvious options would be either to use the union or the intersection of the areas (these are already provided as CombineUnionService and CombineIntersectionService), but you can also define your own combination strategy.

Example 4.4. Service to combine search results

```
public interface CombineResultService {

    /**
     * Combine the envelopes from the match results into the end result.
     *
     * @param results results which need to be combined
     * @return result envelope
     */
    Envelope combine(List<GetLocationResult> results);
}
```

All you have to do is implement the combine() method. The strategy can be configured in the GeocoderInfo instance in your application context.