METACARTA REDHAT GEOTAGGER ADMINISTRATOR'S GUIDE

Last modified on May 20, 2011

Version 4.5

About MetaCarta

MetaCarta, a pioneering geographic information application solutions provider, offers comprehensive analysis tools to enable better business decisions. Founded by a team of MIT researchers in 1999, MetaCarta provides groundbreaking technology that bridges the gap between Geographic Information Systems (GIS) and text search. The company presents innovative solutions that improve decision-making by making it possible for customers to discover, visualize, and act on important information. The company is privately held, with offices in Springfield, OH and Reston, VA. To learn more about MetaCarta, please visit www.metacarta.com.

MetaCarta, a division of Qbase, LLC
250 Veronia Drive, Suite 300
Springfield, OH 45505

Phone: + 1 937 521 4200
Fax: + 1 937 521 2434
Email: info@metacarta.com

Contents

Overview

The MetaCarta GeoTagger and Location Finder for Red Hat Enterprise Linux provides two web services. The GeoTagger geographically tags documents in English and, if licensed, in Arabic, French, Russian, and Spanish. The Location Finder provides geographic locations whose names match a query string. The GeoTagger accepts text documents (ASCII, UTF-8, Latin-1, and some other special encodings) and produces a separate XML result for each file that identifies location references in that document. Each location reference is enclosed in an XML tag that contains confidence information and the latitude and longitude for each location. The GeoTagger does not insert tags into the original document. This approach avoids problems that can arise when the original document is modified, particularly when other tagging systems result in multiple overlapping tags. It also simplifies the integration of GeoTagger output into other systems.

The JSON Location Finder API returns location search results in JavaScript Object Notation, or JSON, format. format. The JSON formatted results are easy to integrate into customized applications, including graphical user interfaces. The JSON Location Finder API searches for locations in installed GDMs and (if used) the custom gazetteer.

The MetaCarta services get their geographic data from one or more Geographic Data Modules, or GDMs. The GeoTagger and Location Finder for RHEL currently only supports the base and foreign language GDMs.

Assumptions

While this document tries to present the administrative interface to the MetaCarta software as simply as possible, a basic level of familiarity with a Red Hat command line interface is assumed. It is also assumed that the administrator has a basic understanding of system administration issues, such as disk space management and directory structure.

The MetaCarta software assumes you are installing on a RHEL Version 5.4 or 4.8 system and are not installing any services on that system other than those the software needs to function.

This document assumes that the DNS name of your system is `metacarta.example.com`.

This document assumes some familiarity with JSON. JSON, or JavaScript Object Notation, is a data-interchange language designed to be easy to use from many languages. More information and tools for working with JSON are available from `http://json.org`.

Installation

Currently, installation on RHEL 5.4 and 4.8 requires different DVDs or ISO images. Please check your installation media to make sure that it matches the version of RHEL on your system. If you are not sure, please contact Customer Support (see page 34).

Initial Installation for RHEL 5.4

To install the GeoTagger and Location Finder for RHEL, mount the software DVD or ISO. In the root directory, for example `/mnt`, you should see three things: `install`, `rpms`, and `srpms`. `install` should be run from the command line as follows:

```
[root@metacarta /]# /mnt/install
```

This will install all required packages. To start the GeoTagger service immediately, run:

```
[root@metacarta /]# /etc/init.d/geo_tagger start
```

To start the Location Finder service immediately, run:

```
[root@metacarta /]# /etc/init.d/location_finder_service start
```

(You will need to follow the instructions in the next section before the services are available over the network.)

Initial Installation for RHEL 4.8

To install the GeoTagger and Location Finder for RHEL, mount the software DVD or ISO. For example, you might run the command:

```
[root@metacarta /]# mount -o loop geotagger-rhel.iso /mnt
```

In the directory `/mnt` (or whatever directory you use on your system), you should see an `rpms` directory. You should enter that directory and install all of the RPMs from the command line as follows:

```
[root@metacarta /]# cd /mnt/rpms
[root@metacarta rpms]# rpm -Uv *.rpm
```

This will install all required packages. To start the GeoTagger service immediately, run:

```
[root@metacarta /]# /etc/init.d/geo_tagger start
```

To start the Location Finder service immediately, run:

```
[root@metacarta /]# /etc/init.d/location_finder_service start
```

(You will need to follow the instructions in the next section before the services are available over the network.)

Initial Configuration

Installing Your License File

The license file is not included on your software DVD, as it is a unique file based on the MAC address of your system. If you do not have a license file, you should contact MetaCarta Customer Support (see page 34) with the MAC address of your eth0 Ethernet interface. You can get this by running `ifconfig eth0`; the "HWAddr" on the first line is your MAC address.

Once you have your license file, copy it to the system, either with `scp` or by pasting it into a new plaintext document. You should then move that file to `/var/lib/metacarta/license`.

Startup Registration

To cause the GeoTagger service to be enabled at boot time, you should run the following command:

```
# chkconfig --add geo_tagger
```

If you ever want to stop the GeoTagger from being enabled at boot, you can use the `ntsysv` tool to change this setting.

To cause the Location Finder service to be enabled at boot time, you should run the following command:

```
# chkconfig --add location_finder_service
```

If you ever want to stop the Location Finder from being enabled at boot, you can use the `ntsysv` tool to change this setting.

Configuring sudo

By default, all of the MetaCarta configuration tools must be run as root. If you prefer, you can configure them to run under sudo for non-root users who are members of the group `metacarta` by adding the following lines to the `/etc/sudoers` file:

```
%metacarta ALL = (ALL) NOPASSWD: /etc/init.d/geo_tagger
%metacarta ALL = (metacarta) NOPASSWD: ALL
```

Confirming Successful Installation

At this point, you should have a fully functional GeoTagger system listening on the localhost interface at port 6050, and Location Finder system listening on the localhost interface on port 6060. You can test this with the following curl commands:

```
# curl localhost:6050/services/GeoTagger/WSDL
# curl localhost:6060/locationfinder --data
"version=2.1.0&query="
```

Sample Apache Configuration for RHEL 5.4

In most cases, you will want to make this service accessible to computers on your network. These instructions show one way to configure the GeoTagger and Location Finder services, using Apache and mod_proxy, making them available on port 80, the default HTTP port. Authentication is outside the scope of this manual; the Apache documentation is available at http://httpd.apache.org/.

If you have not yet installed Apache on the system, you should do so by running `yum install httpd`. Then, in order to let Apache connect to other services, run:

```
# setsebool httpd_can_network_connect true
```

To configure Apache to provide the GeoTagger service and Location Finder service on port 80, you should create the following new file as `/etc/httpd/conf.d/geotagger.conf`:

```
<VirtualHost *:80>
<IfModule mod_proxy.c>
   <Location /services/GeoTagger>
     ProxyPass         http://localhost:6050/services/GeoTagger retry=1
     ProxyPassReverse  http://localhost:6050/services/GeoTagger
     SetEnv            proxy-nokeepalive 1
   </Location>
   <Location /locationfinder>
     <Limit GET>
     order allow,deny
     deny from all
     </Limit>
     ProxyPass         http://localhost:6060/services/location-finder/JSON
       retry=1
     ProxyPassReverse  http://localhost:6060/services/location-finder/JSON
     SetEnv            proxy-nokeepalive 1
   </Location>
</IfModule>
</VirtualHost>
```

If you are only using one of these services, you can leave out the other Location object in the file.

Note: For more information on why GET requests are disabled, please see page 16.

After creating these files, run `service httpd start`.

By default, RHEL has port 80 firewalled, preventing web access to the MetaCarta services. To disable this firewall, run `system-config-securitylevel` and:

- Tab to the "customize" button and press enter.
- Tab to "WWW" and press the space bar to put a * mark there.
- Tab to the "OK" button and press enter.

- Tab to the "OK" button and press enter.

This process will allow incoming connections over port 80. You should now be able to access the GeoTagger WSDL by going to `http://metacarta.example.com/services/GeoTagger/WSDL`; if that URL returns an error, you should check that Apache is running, and check your previous configuration steps.

Sample Apache Configuration for RHEL 4.8

Red Hat Enterprise Linux does not have a standard package distribution system on top of RPM as 5.4 does; we recommend that you install the Apache `httpd` package as you would install other software packages in your local configuration.

To configure Apache to provide the GeoTagger service and Location Finder service on port 80, you should create the following new file as `/etc/httpd/conf.d/geotagger.conf`:

```
<VirtualHost *:80>
<IfModule mod_proxy.c>
   <Location /services/GeoTagger>
     ProxyPass         http://localhost:6050/services/GeoTagger
     ProxyPassReverse  http://localhost:6050/services/GeoTagger
     SetEnv            proxy-nokeepalive 1
   </Location>
   <Location /locationfinder>
     <Limit GET>
       order allow,deny
       deny from all
     </Limit>
     ProxyPass         http://localhost:6060/services/location-finder/JSON
     ProxyPassReverse  http://localhost:6060/services/location-finder/JSON
     SetEnv            proxy-nokeepalive 1
   </Location>
</IfModule>
</VirtualHost>
```

If you are only using one of these services, you can leave out the other Location object in the file.

Note: For more information on why GET requests are disabled, please see page 16.

After creating this file, run `service httpd start`.

By default, RHEL has port 80 firewalled, preventing web access to the MetaCarta services. To disable this firewall, run `system-config-securitylevel` and:

- Tab to the "customize" button and press enter.
- Tab to "WWW" and press the space bar to put a * mark there.
- Tab to the "OK" button and press enter.
- Tab to the "OK" button and press enter.

This process will allow incoming connections over port 80. You should now be able to access the GeoTagger WSDL by going to `http://metacarta.example.com/services/GeoTagger/WSDL`; if that URL returns an error, you should check that Apache is running, and check your previous configuration steps.

Concepts

SOAP

The W3C defines SOAP, the Simple Object Access Protocol, as follows:

> SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics. (`http://www.w3.org/TR/soap12-part1/`)

More simply, SOAP provides a method for wrapping information in XML and then sending it between various middleware packages.

GeoTagger DTD

The GeoTagger DTD (Document Type Definition, the file that defines the schema for a certain type of XML document) is available at `/usr/share/metacarta/geomarkup.dtd`. This DTD specifies how results will be sent back from the GeoTagger. While this manual does not include a complete walkthrough of the DTD, all of the tags that may appear in the current release of the MetaCarta GeoTagger are described on page 14.

Interface Definition

There are five different requests that you can make using the GeoTagger API. Each of them is described here. If you would rather work with examples than read the exact specification of the API, skip to page 30 and return to this section if you need more information on a specific part of the API. If you are not familiar with SOAP, you may be better off working with examples first in order to get a handle on the general format.

GetDocumentCounts

The GeoTagger is licensed to allow you to GeoTag a finite number of documents. To find your license limits and how many documents have been tagged so far, use the GetDocumentCounts request. You can see a sample of the GetDocumentCounts request, and the other requests in this section, in the `/usr/share/doc/metacarta/GeoTaggerExamples.tar.gz` archive. (Instructions for expanding this archive are on page 32.)

This request will return a result like the following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body xmlns:s0="http://metacarta.com/wsdl"
xmlns:s1="http://www.w3.org/2001/XMLSchema">
  <s0:GetDocumentCountsResponse>
    <current xsi:type="s1:long">2</current>
    <maximum xsi:type="s1:long">1000000</maximum>
  </s0:GetDocumentCountsResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `current` value tells you how many documents have already been GeoTagged; the `maximum` value tells you how many documents your GeoTagger can GeoTag before you exceed your license limit. If you need to process more documents than your maximum, you must extend your license; please contact Customer Support (see page 34).

GetSupportedLanguages

This request returns a list of all languages which are currently supported by the GeoTagger. The result will look like the following:

```
  <s0:GetSupportedLanguagesResponse>
    <languages xsi:type="s0:Language">
      <code>arb</code><name>Arabic</name>
    </languages>
    <languages xsi:type="s0:Language">
      <code>eng</code><name>English</name>
    </languages>
    <languages xsi:type="s0:Language">
      <code>fra</code><name>French</name>
    </languages>
    <languages xsi:type="s0:Language">
      <code>rus</code><name>Russian</name>
    </languages>
    <languages xsi:type="s0:Language">
      <code>spa</code><name>Spanish</name>
    </languages>
  </s0:GetSupportedLanguagesResponse>
```

GetSupportedCharacterSets

Thie request returns a list of all character set encodings which are currently supported by the GeoTagger. The result will look like the following:

```
  <s0:GetSupportedCharacterSetsResponse>
    <charsets>ASCII</charsets>
    <charsets>BIG5</charsets>
    <charsets>CNS</charsets>
```



```
    ...
  </s0:GetSupportedCharacterSetsResponse>
```

GeoTagDocument

The standard request is the GeoTagDocument request, which is used to return GeoTags for a document. You can see a sample of the GeoTagDocument request in the `/usr/share/doc/metacarta/GeoTaggerExamples.tar.gz` archive. (Instructions for expanding this archive are on page 32.) A GeoTagDocument request has only one parameter, `document`:

```
<document xsi:type="SOAP-ENC:base64">
SSB1c2VkI...
</document>
```

This parameter must contain the text of the document you want to GeoTag, base64-encoded. The document must be in a plain-text format; HTML is acceptable, but binary documents (like Microsoft Word or Adobe PDF documents) are not. You must convert such documents to text or HTML before sending them for GeoTagging. The GeoTagger will return a set of offset tags — that is, GeoTags that refer to specific points in the document, but are not inserted into the document itself.

(For an example of interpolated or *inline* tagging, see the `geotag` command. Using `geotag --inline` follows the DTD at `/usr/share/metacarta/geoinline.dtd` and not the `geomarkup` DTD in the same directory. We do not recommend inline tagging because it can fail when there are nested tags. For example, the string "Garden Banks Blocks 83, 84" (a way of referring to certain areas in an oil mining operation) might generate tags for "Garden Banks," "Garden Banks Block 83," and "Garden Banks Block 84." There is no way to express these inline.)

GeoTagDocumentWithLanguage

This request adds to GeoTagDocument the ability to specify the language and character set encodings of your documents. (By default, the GeoTagger tries to guess the language and character set of your documents.) A GeoTagDocumentWithLanguage request has five parameters:

```
<element name="fallbackLanguage" type="xsd:string">
<element name="fallbackCharset" type="xsd:string">
<element name="overrideLanguage" type="xsd:boolean">
<element name="overrideCharset" type="xsd:boolean">
<element name="inDocument" type="xsd:base64Binary">
```

`inDocument` is identical to the `document` parameter of GeoTagDocument.
`fallbackLanguage` and `fallbackCharset`, both of type `xsd:string`, allow you to set defaults which will be used if the language or character set encoding of a document cannot be detected. If you would instead like to specify a language or character set that override what the GeoTagger detects in your document, use `overrideLanguage` or `overrideCharset`, both type `xsd:boolean`, along with the corresponding fallback parameters.

To see what languages are available and to see the appropriate abbreviation (for example, eng for English) for each, make a GetSupportedLanguages request; to see what character sets are available, make a GetSupportedCharsets request. A sample request of each type is available in the examples package (see page 30); the results may differ based on what languages you are licensed to tag.

How to Read GeoTagger Results

When you receive GeoTagger results, you will need to process the XML in order to extract the data you want. You can see an example search results package in the `/usr/share/doc/metacarta/GeoTaggerExamples.tar.gz` archive. (To extract this archive, see page 32.) The archives have been base64 decoded for you; when you GeoTag documents yourself, you will need to base64 decode the output before you can read it.

The API will return a `GeoMarkup` object containing zero or more `GeoTag` objects. `GeoTag` objects contain the following information:

```
<GeoTag TagType="GEO" Confidence="0.994366">
  <TextExtent AnchorStart="15" AnchorEnd="23" Anchor="Pasadena">
    <TextComponent AnchorStart="15" AnchorEnd="23" Anchor="Pasadena"/>
  </TextExtent>
  ...
```

The `Confidence` is the GeoTagger's confidence that the word is geographic in its context in the document, expressed as a value from 0 (not geographic) to 1 (definitely geographic). Place names can often also be the names of persons, organizations, and products .

The `TextExtent` object contains one or more `TextComponents`, which are strings of geographic text (in this case, "Pasadena") and the locations those strings appear in the document. In this case, the word "Pasadena" started on the 18th character of the document and ended on the 26th. Many `TextExtents` will have only one `TextComponent`, but street addresses (when you have an optional Street Address GDM installed) and similar constructions may contain more than one.

Note: `AnchorStart` and `AnchorEnd` refer specifically to the number of characters, not bytes. A multi-byte character will still only count as one character for the purposes of this measurement.

```
<Disjunct Weight="0.392758" FeatureType="DOT" GazetteerID=
  "MetaCarta Gazetteer v4.5.0">
  <Conjunct Name="Pasadena" Class="P" Type="PPL" Population="141674"
    Country="US" CountryName="United States" CountryConfidence="0.381427"
    Province="US48" ProvinceName="Texas" ProvinceConfidence="0.381427">
    <Dot Latitude="29.6911" Longitude="-95.2091" DotWeight="1.0"/>
  </Conjunct>
</Disjunct>
<Disjunct Weight="0.392758" FeatureType="DOT" GazetteerID=
  "MetaCarta Gazetteer v4.5.0">
  <Conjunct Name="Pasadena" Class="P" Type="PPL" Population="133936"
    Country="US" CountryName="United States" CountryConfidence="0.381427"
    Province="US06" ProvinceName="California" ProvinceConfidence="0.381427">
```



```
    <Dot Latitude="34.1478" Longitude="-118.145" DotWeight="1.0"/>
  </Conjunct>
</Disjunct>
...
```

Each GeoTag will contain one or more `Disjuncts`, potential places to which the text extent might refer. In this case, two disjuncts are given with different `Weight`, the percentage likelihood that the disjunct refers to the correct location. The `FeatureType` will always be "DOT" in this release, signifying that the location is modeled as a point.

Each disjunct contains one `Conjunct`, which expresses the latitude and longitude of the associated disjunct, as well as enclosing country and province information. If you have looked at the DTD, you will notice that the GeoTagger specification allows for more than one Conjunct per Disjunct, for cases where a location might be expressed in multiple ways. However, in this release, each disjunct will contain exactly one conjunct.

The conjunct `Name` is the canonical name, according to the MetaCarta GDM, of the location. This will often, but not always, be the same as the original text in the document. Each conjunct may have a `Class` and a `Type` expressing the class and type of geographic location represented by the latitude and longitude of the conjunct. The nine potential classes are as follows:

P	Populated place
H	Hydrographic
T	Hypsographic
S	Spot feature
L	Locality or Area
A	Administrative region
V	Vegetation
U	Undersea
R	Streets, highways, roads, or railroad

There are 643 different types; for a complete list of types, please see the file `ConjunctTypes.txt` in the examples archive.

To see a list of potential `Country` and `Province` codes, please see `/usr/share/doc/metacarta/fips_country_province_codes.txt`.

The `CountryConfidence` and `ProvinceConfidence` express the GeoTagger's confidence that the correct country and province have been chosen for this place name.

If you are tagging a document in Arabic, the place names will be returned in Arabic inside the XML results. The names will be properly formatted for right to left display, but may not display properly in some editors. Both Arabic and Russian tags will be returned in UTF-8. For example:


```
<TextComponent AnchorStart="7" AnchorEnd="13" Anchor=""/>

<TextComponent AnchorStart="55" AnchorEnd="61" Anchor="Мьянме"/>
```

French and Spanish placenames that contain non-ASCII characters will also be properly formatted:

```
<TextComponent AnchorStart="1377" AnchorEnd="1382" Anchor="París"/>
```

Communicating with the Location Finder

The JSON Location Finder API allows you to search for locations from the Geographic Data Modules installed on your system. The search request takes the form of a URL; results are returned in JSON format.

API Requests

You can request information from the JSON Location Finder API by constructing a request URL with appropriate parameters as listed in the following sections. A request URL has two parts. The first part is the service's base URL on your system followed by a question mark. The second part is a list of parameter *name=value* pairs separated by & (the ampersand). For example, here is a JSON Location Finder API URL containing the API version parameter and a query for the place "Boston":

```
http://metacarta.example.com/locationfinder?version=2.0.0&query=boston
```

Any parameter that you do not specify takes on its default value. If your input does not make sense, the service will return an error message. For example, if you pass in an alphabetic character in a parameter that can only be an integer, the service will return an error. In some cases, these error messages can be used to determine the cause of failure for the query.

The JSON Location Finder API uses HTTP as its transport protocol; you can use any HTTP client to interact with the API. All modern programming languages have HTTP client libraries, most of which allow you to set HTTP Request Headers. If you send your request with an an 'Accept-Encoding' HTTP Request Header that contains 'gzip', the JSON API will compress the output before sending it over the wire to you. (The HTTP return response will contain a Response Header called 'Content-Encoding' with the value 'gzip' indicating to the client that it should uncompress the response content.) Compression can significantly decrease the time that a client must wait to retrieve content. Many web browsers set this header automatically, helping performance in AJAX applications.

JSON Security

When a trusted person's browser can access both a JSON callback API that offers secured content and a host operated by an untrusted party, there is potential for attack. Specifically, if the trusted person loads a page from the untrusted host, that page could cause the browser to load information from the JSON callback API and

then subsequently pass that information to the untrusted host. This would allow an untrusted party to access GTS search results and other JSON API output using the trusted party's credentials, potentially violating security policies.

There are two risk scenarios. The first involves information that is not subject to access control, and the second involves information that is subject to access control.

In the first scenario, the JSON service is available to all users within a private network. The information indexed by the service is not, however, intended to flow beyond a certain private network. If a trusted person operating a browser in the network browses to an attacker's website outside of that network, that website can serve the trusted user's browser a page that loads information from the service and siphons it outside of the trusted network.

Networks created by a firewall or Network Address Translation (NAT) device are especially vulnerable to this type of attack.

In the second scenario, the service contains secured content which is only accessible to individuals with appropriate security credentials. When a trusted user makes HTTP GET requests to the service, the requesting browser packages the user's credentials with the request. If the trusted user then browses to an attacker's website either inside or outside of a private network, that attacker's website can serve a page into the trusted user's browser that can load information from the service using the trusted person's credentials. This would allow the attacker, who could be located inside or outside of the trusted network, to access documents that should not be accessible.

Networks where different users have different access permissions are vulnerable to this type of attack, even if those networks have no connections to the outside world.

The attack poses no threat to JSON callback APIs serving information that can be read by every person with physical access to any part of the network (within or beyond firewalls). For example, the attack poses no major threat to a host on the public Internet that only serves information that anyone on the public Internet can see. In this case, all an attacker could do is make requests to the service that appeared to be coming from another user.

This risk only exists when using GET requests to the JSON API, not POST requests. Therefore, by default, the service is configured with GET requests to the JSON Location Finder API turned off entirely. (The Location Finder can contain confidential information in the form of your custom gazetteer.) You can enable GET requests by removing the Limit stanza from the Apache configuration file you configured earlier.

How to Read JSON Results

JSON has two types of multi-part data structures: *arrays* (also known as lists) and *objects* (also known as *dictionaries* or *hashes*). An array is bracketed by square braces:

```
[ listItem1, listItem2, listItem3 ]
```

An object (or *dictionary*) is bracketed by curly braces:


```
{
  dictKey1:  dictValue1,
  dictKey2:  dictValue2,
  dictKey3:  dictValue3
}
```

In *key*:*value* pairs, the key must be a string. The value can be any of seven types: string, number, object, array, boolean true, boolean false, null.

The JSON Location Finder API

The JSON Location Finder allows you to conduct searches for locations from GDMs installed on your appliance. Each search returns a list of locations matching your request parameters, as well as information summarizing the search and its results.

Location Finder URLs and Requests

You can request information from the JSON Location Finder API by constructing a URL including any of the parameters specified in the Location Finder parameter list below. Here is an example JSON Location Finder API URL request that queries for locations named "Elephant" :

http://metacarta.example.com/locationfinder/JSON?version=2.0.0&query=elephant

Additionally, the Location Finder provides a reverse lookup feature which allows the user to query for all the possible locations found in a specified geographic region. The user can specify the interested geographic region in terms of a bounding box or a centroid with radius.Here is an example JSON Location Finder API URL request that queries for locations within a given region specified by a bounding box (for example with bounding box being "24.988187,13.320807,25.559471,13.848146"):

http://metacarta.example.com/locationfinder/JSON?version=2.0.0&bbox=24.988187,13.320807,25.559471,13.848146

Here is an example JSON Location Finder API URL request that queries for locations within a given region specified by a centroid with radius (for example with centroid being "25.322580,13.596150" and radius being "30" kilometers):

http://metacarta.example.com/locationfinder/JSON?version=2.0.0&centroid=25.322580,13.596150&rad=30

It should be noted that unlike the query string match feature, which has got access to the complete base and all the possible add-on GDM locations , the reverse lookup feature is limited to US government published locations (USGS and NGA) and custom Gaz locations only.

Location Finder Parameters

The following sections describe the parameters that can be used in JSON Location Finder API requests. The Location Finder is characterized by one important parameter.

Major Location Finder Parameters: Place Name and Reverse Lookup Region

Place Name

Place name is the primary search parameter for location searches. A Location Finder request incorporates place name and administrative district data through the query parameter.

To search for a location, enter the place name of that location. The GDMs contain place name data for a wide range of location types, from towns and landmarks to countries and continents. To create more specific searches, include containing administrative location name data in a comma-separated list. For example, a location search for "Cambridge" returns approximately 75 location results, from Cambridge in the United Kingdom to Cambridge, Texas. To restrict the location search to the United States, you could search for "Cambridge, USA", which in turn reduces the number of results to about 45. Searching for "Cambridge, Massachusetts" or "Cambridge, MA, USA" yields only one result; the location named Cambridge whose administrative path is "Cambridge, Middlesex, Massachusetts, United States".

The API assumes that commas in a query string are used to delimit parts of the administrative path. If you are trying to search for a place with a comma in any portion of its name or the name of any part of its adminitrative path, the comma in the name should be represented by a URL escaped comma, `%2C`. For example, when searching for "Boston, city of", the query key should be `boston%2C city of`. Spaces can also be represented with the URL coding for a space, `%20` or by a plus (`+`).

Some clientsalso need the percent sign escaped because they do not URL-encode the query itself. In these clients, commas must be entered as %252C if you do not want them to delimit parts of the administrative path. The sample client in the examples package (see page 30) does URL-encode the query, so %2C is sufficient.

Reverse Lookup Region

Reverse Lookup region is the primary search parameter for querying the location finder for all possible locations present in a specified geographic region. A Location Finder request incorporates reverse lookup region through the bbox and centroid with rad parameters.

The bbox parameter can be specified as a comma separated list of coordinates for a bounding box encompassing the region of interest. Further, the specified bounding box coordinates should be in this order : Minimum Longitude, Minimum Latitude, Maximum Longitude, Maximum Latitude (I.E. the lower left coordinates followed by the upper right coordinates of the bounding box, in signed decimal coordinates without a direction context).

The alternative way to specify a geographic region of interest is by a centroid with the rad (radius) parameter. The specified centroid coordinates should be in this order : Longitude, Latitude (I.E Longitude followed by a Latitude of the centroid point, in signed decimal coordinates without direction context) and the rad (radius) should be specified in kilometers.

It is suggested that the geographic extent of the specified reverse lookup region be less than 100 kilometers for regions with high populations. By default the reverse lookup search is limited to a maximum radius of 100 kilometers, but the user can override the extent as mentioned later in this section.

Further, it is important to note that the "query", "bbox" and "centroid" (with "rad") are mutually exclusive parameters for the Location Finder and combinations of any two of them or all specified together will be treated as a an invalid input parameter combination.

Location Finder Parameter List

The following parameters can be used with the Location Finder API:

- `version` (Required)
  Default value: None
  Allowed values: `2.0.0, 1.1.0`
  Usage: The requestor must specify the API version they wish to use. Although the JSON Explorer sets the version=2.0.0 by default, the API requires a valid version parameter.
- `query` (Required / Mutually Exclusive with "bbox" and "centroid")
  Default value: (blank)
  Allowed values: `string`
  Usage: The location string being searched for. Administrative districts should be separated by commas, for example "Cambridge, MA, USA."
- `bbox` (Required / Mutually Exclusive with "query" and "centroid")
  Default value: (blank)
  Allowed values: `minlon,minlat,maxlon,maxlat`
  Usage: A comma seperated coordinate list representing the reverse lookup region.
- `centroid` (Required / Mutually Exclusive with "query" and "bbox")
  Default value: (blank)
  Allowed values: `lon,lat`
  Usage: A comma seperated coordinates representing the centroid of the reverse lookup region , whose extent is specified by the "rad" parameter.
- `rad` (Optional / Required with "centroid")
  Default value: (blank)
  Allowed values: `distance`
  Usage: A radius of the reverse lookup region in killometres.
- `ovmr` (Optional)
  Default value: None
  Allowed values: `1`
  Usage: Setting ovmr to "1" allows user to overide the maximum default radius (I.E. the "500 KilloMetres") for the reverse lookup region.
- `geometry` (Optional)
  Default value: None
  Allowed values: `WKT`
  Usage: Setting the geometry format for a location finder query will return the geometry in that format for any locations for which it is available.
- `callback` (Optional)
  Default value: None
  Allowed values: string
  Usage: The 'callback' parameter facilitates the use of the 'script-adding' technique to load data from

the GSRP, subverting the browser's cross-domain limitation. The user adds a new script tag to the current HTML document, whose url is the full request, with a function name specified as the 'callback' parameter. The GSRP's server response is simply the calling of the specified callback function, with the results object as the first parameter.

JSON Location Finder Result Fields

When you send a Location Finder request, the service will return a JSON object containing the results of your location search.

Just as in a document search request, the output of a successful location search contains three general categories of results: input query parameters, system parameters, and result locations and other results parameters. While there are fewer input query and system parameters, they serve similar purposes as they do in a search request.

- `Query` is a string showing the query you sent to the API.
- `Bbox` is a string showing the value of "bbox" you sent to the API.
- `Centroid` is a string showing the value of "Centroid" you sent to the API.
- `Radius` is a string showing the value of "rad" you sent to the API.
- `OverrideMaxRadius` is a string showing the value of "ovmr" you sent to the API.
- `RegionReference` is a hashed reference to region data for your query that is stored on the API. This output parameter should be ignored; it is currently only used internally.
- `SystemVersion` is the MetaCarta software version and API version of the service with which you are communicating.
- `Styles` is a dictionary containing graphical representation information for each result style. In the RHEL Location Finder, this URL is currently only a placeholder for where the URL would be on a MetaCarta appliance; we recommend that you provide your own location icon.

The bulk of the results will be result location listings. Each location listing contains location data including location geometry, population, and type. The JSON output also includes output parameters that provide an overview of the location results. This includes a dictionary of location types, giving both long and short descriptions of the types contained in the results.

- `Warnings` is a list of warnings (as strings) resulting from your search.
- `Errors` is a list of errors (as dicts) resulting from your search. For example:

```
u'Errors': [u'Message': u'unable to connect', u'Code': None]
```

- `ResultsCreationTime` is a human-readable string stating the date and time that this result set was generated.
- `Attribution` is a human-readable string stating the copyright information for the location result set.
- `ViewBox` is a dictionary of values containing the minimum and maximum latitude and longitude of a bounding box that covers all the returned locations.

- `Types` is a dictionary providing extended descriptions for the 'Type' codes given for the set of location results. The entry for each 'Type' code gives a 'ShortDescription' and a 'LongDescription'.
- `Locations` is a list of Location objects containing the following components:
  - `Paths` is an object that consists of containing information for the location. It will include the value 'Administrative,' which gives administrative district data for the location.
  - `Style` is a string identifying the style that is best used to display this reference in a map. This string is designed to be used as a key in the 'Styles' list included in the results.
  - `Name` is the name of this location.
  - `Geometry` is a dictionary of different representations of the location's geometry.
  - `LocID` is a non-human-readable string that uniquely identifies a tagged location.
  - `Centroid` is an object containing `Latitude` and `Longitude`, which are floating point numbers indicating a center point of the location in unprojected coordinates.
  - `FIPSCode` is the Federal Information Processing Standards Code assigned to the location, if any. For more information on FIPS Codes, please see `/usr/share/doc/metacarta/fips_country_province_codes.txt` .
  - `Type` is a short string representing the type of the location. This string is designed to be used as a key in the 'Types' list included in the results.
  - `Class` is a single character representing the type of the location with less precision than Type.
  - `ViewBox` is a dictionary of values containing the minimum and maximum latitude and longitude of a minimal bounding box containing the given location.
  - `Population` is the integer representing the estimated population of a location as gathered from census data or other sources.
  - `Distance` is the decimal value representing the calculated distance of a location from the centroid of the reverse lookup region. Please ignore this field for location search with "query" command.
  - `Azimuth` is the decimal value representing the calculated azimuth angle of a location with respect to the centroid of the reverse lookup region. Please ignore this field for location search with "query" command.

A sample Location Finder output is included in the Examples Archive. See page 32 for instructions on extracting and accessing the archive.

See the Examples section this document (starting on page 30) for laguage-specific examples on producing results.

The GeoTagger Command Line Tool

The GeoTagger for RHEL package includes the command-line tool `geotag`, useful for testing the GeoTagger and certain types of batch jobs. `geotag` is used much like the standard UNIX `mv` and `cp` commands. The standard command invocation is as follows:

`user@metacarta:~$ geotag` ⟨file1⟩ `[`⟨file2⟩ `...]` ⟨destination⟩

Files must be in a plain text format like .txt or HTML.

If there is only one file to be geotagged, the destination may be either a file (which will be created containing the XML stand-off tags) or a directory; if there is more than one file to be geotagged, the destination must be a directory, and the new files will be created with names in the form `file1.geotag.xml`.

If you pass only one argument to `geotag`, then `geotag` will tag the file described in the argument, displaying the tags on STDOUT. If no arguments whatsoever are passed, then `geotag` will act as a filter, tagging STDIN to STDOUT.

There are a number of additional options to `geotag` that can be set as command-line flags. You can see all of them with brief explanations by running `geotag --help`; explanations of a few of the more common commands are below.

- `geotag --inline` will return an XML document containing the original text of the source with inline GeoTags conforming to `/usr/share/metacarta/geoinline.dtd`, a deprecated format. This approach occasionally causes problems in documents with overlapping geographic references.

  Note: This option requires exclusively UTF8 input.

- `geotag --replace-non-ascii` replaces all non-ASCII text in the original document with placeholder ASCII characters.
- `geotag --continue-on-error` will continue producing tags even if there is an error in one file. By default, `geotag` will stop processing all documents on an error.
- `geotag --nowarn` will ignore dropped tags and not print warnings to the console. This flag is only necessary when using the `--inline` option, which sometimes drops tags on documents with overlapping geographic references.
- `geotag --supported-languages` prints a list of all languages supported by the GeoTagger and then exits. This list includes the three-letter language codes used by other `geotag` arguments.
- `geotag --supported-charsets` prints a list of all character set encodings supported by the GeoTagger and then exits.
- `geotag --fallback-language` ⟨language_abbreviation⟩ will assume the document is written in this ⟨language_abbreviation⟩ if its language cannot be detected. The geotagger recognizes the abbreviations specified in ISO 639-3 (see http://www.sil.org/iso639-3/). To specify that the geotagger should use this language even if a different one is detected, use `geotag --fallback-language=`⟨language_abbreviation⟩ `--override-language`.
- `geotag --fallback-charset=`⟨charset⟩ will assume the document is encoded in the character set ⟨charset⟩ if its character set cannot be detected. To specify that the geotagger should use this character set even if a different one is detected, use `geotag --fallback-charset=`⟨charset⟩ `--override-charset`.
- `geotag --counts` will return the number of documents geotagged and the number of documents that you are licensed to geotag.

Example GeoTagger Usage

To tag a simple string, simply enter the following:

```
metacarta:~$ echo "MetaCarta is located in Cambridge, MA." |
geotag
```

To GeoTag multiple documents in place:

1. Make a directory in which to store the files to be GeoTagged:

   ```
   metacarta:~$ mkdir /data/docs
   ```

2. Make a directory in which to store the GeoTagged documents:

   ```
   metacarta:~$ mkdir /data/geotag
   ```

3. Copy the files to be GeoTagged to the directory that you just created:

   ```
   metacarta:~$ scp myhost:news/* /data/docs
   ```

4. GeoTag the documents:

   ```
   metacarta:~$ geotag /data/docs/* /data/geotag
   processing file:  /data/docs/News_20030415.txt
   processing file:  /data/docs/News_20030413.txt
   processing file:  /data/docs/News_20030412.txt
   ```

   If you have a very large number of documents (more than 1,000) in that directory, we recommend the following invocation instead:

   ```
   metacarta:~$ find /data/docs | xargs -i geotag {}
     /data/geotag
   ```

5. View the GeoTagged documents:

   ```
   metacarta:~$ less
   /data/geotag/News_20030412.txt.geotag.xml
   ```

Customization

To tune GeoTagger behavior, you can edit parameters in with the `geotagger_control` command. The GeoTagger will not notice these changes until the next time it starts up, so that running this command will not interrupt currently running GeoTagger jobs. Restart the GeoTagger by running

```
geotagger_control restart
```

The rest of this section documents the parameters that can be changed via the `geotagger_control` command.

Adjusting GeoTagger Minimum Confidence

The MetaCarta GeoTagger defaults to tagging nearly every possible location reference, including those with very low confidence. Increasing the minimum confidence parameter will cause the GeoTagger to ignore references with confidences below the value of this parameter. This parameter's value must be between 0.0001 and 1, inclusive.

```
metacarta:~$ geotagger_control set
minimum_geo_tagged_geographic_confidence=0.10
```

This example shows how you would set your minimum confidence to 0.10; the default value is 0.05.

Adjusting GeoTagger Points Tagged per Georeference

By default, the GeoTagger returns all matching feature points associated with a tagged geographic reference. You can adjust this behavior so that only one point is returned with each tagged georeference by running the following command:

```
metacarta:~$ geotagger_control set
multiple_points_per_tag=false
```

To restore the default behavior, change this value to 'true.'

Controlling Geographic Extraction of Headers in MIL-STD-6040 documents

The GeoTagger defaults to tagging entire documents. However, in many cases, users do not want to GeoTag message header information. The `geo_extract_message_header` parameter allows you to tell the system to ignore header information on MIL-STD-6040 documents. Set this value to 'true' to extract geographic information from the headers of MIL-STD-6040 documents. Set this value to 'false' to disable extracting geographic information from the headers of MIL-STD-6040 documents. For more information about MIL-STD-6040, also known as USMTF, see http://www.sec.army.mil/did/usmtf_std.html.

```
metacarta:~$ geotagger_control set
geo_extract_message_header=true
```

This example shows how you would turn on header processing. To turn header processing back off, you can reset this value back to 'true.'

Controlling Extraction and Partial Resolution of Coordinates with no Direction Context

By default, the GSRP appliance will not detect or resolve any decimal degree coordinates without a valid direction context. Strings such as "34.987, -123.678" are not treated as geographic references as they would be if they were anchored with a valid direction context such as "34.987N, 123.68W"). To enable such strings to be treated as geographic content, run the following command:


```
metacarta:~$ geotagger_control set geo_tag_no_ref_coords=true
```

The above command will enable the extraction and resolution of coordinates with no direction context. To disable or to restore the default behavior, change this value to 'false.'

Status and Defaults

To see the status of all parameters, run the following command:

```
metacarta:~$ geotagger_control show
parameter name                              value
------------------------------------------ -----
geo_extract_message_header                  false
minimum_geo_tagged_geographic_confidence  0.05
multiple_points_per_tag                    true
```

To reset all parameters to default, run:

```
metacarta:~$ geotagger_control reset all
```

You can also reset individual parameters by replacing "all" with the name of the parameter to be reset.

Adding Entries to the Custom Gazetteer

The MetaCarta GDMs contain a tremendous number of place names and geographic coordinate formats. However, if you wish to have MetaCarta tag a specialized list of names corresponding to latitude-longitude coordinates that you provide, you can add up to four million entries to the custom gazetteer. Most users create this list on another machine and then copy the file. One common use case for the custom gazetteer is tagging military Basic Encyclopedia Numbers, also known as BE Numbers. Locations added to the custom gazetteer are used for geotagging documents through the GeoTagger and displaying locations in the Location Finder.

One downside to adding names to the gazetteer is that the system interprets the list without as much context as those names in the MetaCarta GDMs — if you add the place name "the" to the list, then every instance of "the" in searchable documents could potentially be tagged as if the latitude and longitude you entered were intended by the author to be associated with the word "the." Because of this potential for problems, we recommend that you speak to Customer Support (see page 34) before implementing a custom gazetteer, and make use of the optional quality parameter in the custom gazetteer format.

Gazetteer File Format Rules

To cope with the wide variety of available character sets and unusual word formats, MetaCarta uses a specialized format for adding new data to the custom gazetteer. This format structure is described here. If you feel that this removes important

information about your locations, please contact Customer Support (see page 34) to discuss alternatives.

The gazetteer additions must be formatted as follows:

- Each entry is on a separate line.
- Each line adding a new place name is formatted as:

  ⟨place name⟩ | ⟨latitude⟩ | ⟨longitude⟩ | `1` | ⟨optional-quality⟩ | ⟨optional-enclosing-region⟩ | ⟨optional-location-type⟩

  Each line removing a place name is formatted as:

  ⟨place name⟩ | `REMOVE`

  if you wish to instead disable a name already in the MetaCarta GDM. You can also disable just an individual point. To do so, get the exact latitude, longitude, enclosing region (if any), and location type (if any) from either the Location Finder or the GeoTagger. Then write a line like this:

  ⟨place name⟩ | `REMOVE` | ⟨latitude⟩ | ⟨longitude⟩ | ⟨enclosing-region⟩ | ⟨location-type⟩

  This will remove just that one point by that name. After removing a point or an entire name in one of the two ways shown above, you can then add additional points by that name.

  Note: Removing a location from the custom gazetteer does not remove it from the Location Finder, it only prevents it from being geotagged.

  The enclosing region must be the most specific two or four letter code, if any, in the Paths object; if there is no enclosing region, leave that field empty.
- All fields must be separated by the pipe character (|).
- Commented lines must be preceded by the # character; if you wish to enter a # in a placename, you should double it like this: ## .
- The place name is the word or phrase you wish to match in text.
- Spacing is important:
  - Anywhere multiple spaces adjoin, collapse the multiple spaces down to a single space.
  - Remove any space that appears before or after a place name. The place name should be immediately followed by the pipe character.
- The latitude and longitude must be unprojected in the WGS84 datum and written as floating point values (latitude between -90.0 and 90.0, and longitude between -180.0 and 180.0). Use a leading '-' for south of the equator or west of the prime meridian. These values will be resolved to the nearest millionth: 34.000001 and 34.000001111 will both be treated as 34.000001.
- The optional quality field can contain one of the following three values:
  - `custom-gaz-bad`: To be used for values that are only rarely geographic (for example, Media or Brittany).
  - `custom-gaz-avg`: To be used for values that are sometimes geographic but sometimes not (for example, Washington).
  - `custom-gaz-good`: To be used for values that are almost always geographic (for example, BE numbers).

This value is just a hint to the system; if you do not enter a value here, the system will assume `custom-gaz-avg`.

- The optional enclosing region should contain the code for a country or province that encloses the location you have entered. If there is no appropriate region, just leave this field blank. When removing an existing location, the enclosing region must be the most specific two or four letter code, if any, in the Paths object in Location Finder output or in the Conjunct object in GeoTagger output. To see a list of acceptable codes, please see the file `/usr/share/doc/metacarta/fips_country_province_codes.txt`.
- The optional location type should contain the code for the type of location the point is: building, city, well, and so on. To get the correct code, please see the list in the file `ConjunctTypes.txt` available in the GeoTagger documentation archive: `/usr/share/doc/metacarta/GeoTaggerExamples.tgz`.
- The format of the file should be UTF-8; an ASCII (plain text) file is acceptable, but if your location names include characters that cannot be expressed in ASCII, use UTF-8. You can convert documents into UTF-8 with the `iconv` command. For more information on this command, run `man iconv` from the command line.
- If you have questions about this, or want tools to make this conversion process easier, please contact Customer Support (see page 34).

Gazetteer Format Example

North Truro, a city in Massachusetts which is at 42 degrees north, 70 degrees west, would be entered as:

```
North Truro|42.0|-70.0|1|custom-gaz-good|US25|PPL
```

To remove London, the city in England, from the Custom Gazetteer without affecting other places named London, you would enter:

```
London|REMOVE|51.5|-0.116667|UKH9|PPLC
```

Limitations of the Custom Gazetteer

While the custom gazetteer is a valuable tool for adding locations to the MetaCarta GDM, a location added in with the custom gazetteer does not have the learned linguistic and statistical information associated with it that a location added by MetaCarta does. In some cases, this can lead to lower quality results:

- When multiple locations have the same name and different coordinates, if all of them are custom gazetteer coordinates, the large number of possible locations will reduce the confidence our system has in any single option. This could cause all of the points to fall below the threshold for appearing in search index results.
- When multiple locations have the same name and different coordinates, and some of them are custom gazetteer coordinates, the point in the MetaCarta GDM can sometimes obscure the custom gazetteer coordinates, or vice versa. For example, if a number of custom gazetteer points are named "New York,"

they are unlikely to be tagged in place of New York City given what the GDM knows about its prominence. On the other hand, if an uncommon name for a much less prominent location in the GDM is competing with a number of custom gazetteer points, it is possible that the custom gazetteer points will replace it in results completely.

- When multiple locations have different names but identical coordinates, the GeoTagger will tag all of them, but, if the index is not set to extract multiple points per tag, search index results will only tag the highest relevance point for any exact latitude and longitude. In the case of a tie, which is likely if all of the points are in the custom gazetteer, the first one in the list will be tagged.

Verifying the Custom Gazetteer

Run `custom_gazetteer_control verify` ⟨filename⟩ to verify that the file format of the gazetteer you created is correct:

`custom_gazetteer_control verify` ⟨filename⟩

If you see an error like this one:

```
7,000,000 custom gazetteer entries, which is more than the
maximum of 4,000,000
```

Your custom gazetteer contains too many entries. Both additions and removals count as individual entries.

If you see an error like this one:

```
MCERR build_custom_gazetteer.cpp:283 [int main(int, char**)]:
/home/mcadmin/customgaz:3:  failed to remove entry [Laaaa]
that was not in the MetaCarta GDM
```

You are trying to disable a term that does not appear in the MetaCarta GDM. The term, in this example, is on line 3 of your custom gazetteer file; remove that line from your gazetteer and then try again. If you continue to have problems verifying your custom gazetteer, contact Customer Support (see page 34).

Installing the Custom Gazetteer

Run `custom_gazetteer_control install` ⟨filename⟩ to install the gazetteer you created onto the system:

`custom_gazetteer_control install` ⟨filename⟩

Displaying the Custom Gazetteer

You may want to see what locations are currently in your custom gazetteer; the file you edit and install is only used to *create* the gazetteer. To do so, run `custom_gazetteer_control show` from the command line. If you want to output the gazetteer data to a file, which may be helpful if your gazetteer is very large, append a filename to the end of that command.

Removing the Custom Gazetteer

Run `custom_gazetteer_control remove` to remove the gazetteer you created from the system. This will not delete your custom gazetteer file; the system will simply ignore the file.

Examples

Appliance Name and Authentication for All Examples

All of the following examples assume that HTTP Basic Auth has been configured for a user named `fred` with a password of `ginger`. This document does not explain how to set HTTP Basic Auth or other security on the GeoTagger service.

curl

`curl` is a command line utility available for both Linux and Windows from `http://curl.haxx.se`. Since `curl` is an HTTP client and can set custom headers, you can use it to send GeoTag requests if you already have valid XML requests in text format. For example, to send the request `request.txt` to the GeoTagger API for `metacarta.example.com`, run the following command:

Note: The following command should be on one line; it is line-wrapped in this example for your convenience.

```
metacarta:~$ curl -d "@request.txt" -H "Content-Type:  text/xml"
  -H "SOAPAction:\"\""
  http://fred:ginger@metacarta.example.com/services/GeoTagger
  > output.txt
```

If you want to redirect the output to another application, you can replace `>output.txt` with `| processing_command`.

A number of example requests are included in the archive `/usr/share/doc/metacarta/GeoTaggerExamples.tgz`. These requests and responses are included:

- `GetDocumentCountsRequest` is a sample XML request that could be sent via curl to get the number of documents that have been GeoTagged and the GeoTagger's license limit.
- `GetDocumentCountsResponse` is a sample XML response to the request above.
- `polio.txt` is a sample document to GeoTag.
- `GeoTagDocumentRequest` is a sample XML request that could be sent via curl to get GeoTags for the document above.
- `GeoTagDocumentResponse` is a sample XML response to the request above. It has been base64 decoded.
- `GetSupportedLanguagesRequest` is a sample XML request that could be sent via curl to get the list of supported languages.

- `GetSupportedLanguagesResponse` is a sample XML response to the request above.
- `GetSupportedCharacterSetsRequest` is a sample XML request that could be sent via curl to get the list of supported character sets.
- `GetSupportedCharacterSetsResponse` is a sample XML response to the request above.
- `SampleDocuments` contains sample documents in different languages.
- `SampleDocumentGeoTags` contains GeoTagger output for the documents in the directory above. They have been base64 decoded.

Instructions for accessing these files are on page 32.

Java

An example package is included with this document in the examples archive. The `JavaGeoTaggerExample` directory in the archive (instructions for accessing the archive are on page 32) contains the following:

- `GeoTaggerClientExample.java`, code used to access the GeoTagger API
- A `Makefile`, used to build and run the program
- A `sample.input` file, containing a document that the example will GeoTag when you compile and run it

When you have extracted these files, running `make` at the command line will compile `GeoTaggerClientExample.class` for you, and `make run` will run the example and produce a `sample.output` file for you. If you are running RHEL 5.4, you will need to install additional packages in order to use this example; the following command:

```
# yum install -y java axis wsdl4j log4j
java-1.6.0-openjdk-devel
```

Running `make run` will only work if you have configured Apache to provide the GeoTagger service on port 80. If you choose not to do this, you can instead test this example by running `./run` with `--host localhost:6050` added to the list of options.

If you are running RHEL 4.8, there is no running Java example available.

Python

An example script is included with this document in the examples archive. `geotagger.py` is a simple script that takes two arguments, a text document and the DNS name of a GeoTagger, and returns the XML GeoTags provided by the API. To use it from the command line, run:

```
./geotagger.py sample.html metacarta.example.com
```


This script uses the `suds` package. If running the code on another machine, you will need to install this package from another source, or change the code to use a different SOAP package of your choice. For more information on suds , please see `http://fedorahosted.org/suds/`. If you are running RHEL 4.8, you will also need to use Python 2.4 as provided by the GeoTagger software package:

```
/usr/bin/python2.4 geotagger.py sample.html
metacarta.example.com
```

C/C++

Using C/C++, you have access to the library version of `curl` (see page 30). You may also wish to use SOAP middleware such as Systinet Server.

Since the HTTP protocol is so simple, and only basic requests and responses must be made, developing directly against the operating system's sockets API is also an option.

Please contact Customer Support (see page 34) for more help with the SOAP Search API using this language.

Extracting the Examples Archive

To extract the examples archive to a new directory, `GeoTaggerExamples`, in the current working directory, simply run the following command:

```
tar -xvzf /usr/share/doc/metacarta/GeoTaggerExamples.tgz
```

You may find it convenient to just extract the entire archive into the `/usr/share/doc/metacarta/` directory. This will allow you to access all of the attached files at any time without having to expand them, but does take up additional disk space.

Location Finder Examples

There is an additional package containing two Location Finder examples in the file `/usr/share/doc/metacarta/LocationFinderExamples.tgz`. The files it contains are:

- `JSONLocationFinderClient.py` : A Python client for the JSON Location Finder.
- `locationfinderoutput.txt` : Sample JSON Location Finder output.

Troubleshooting

The GeoTagger and Location Finder services log to `/var/log/messages`.

- If you are experiencing performance problems, first try the `geotag` command from the command line. That application will not suffer from middleware or network latency. If `geotag` is still slow, please contact Customer Support (see page 34).

- If you see the following error: "Unicode replacement character (0xFFFD) found in input UTF-8 string at byte X, indicating conversion or character set problems," your document contains a Unicode replacement character, which generally means that the document was converted incorrectly and is unreadable. If the document is otherwise correct, remove the 0xFFFD character(s) and try sending it again.
- In response to a GeoTag request, you may get an error informing you that the language of that document is unlicensed. If you would like to purchase a GeoTagger license for that language, please contact Customer Support (see page 34) or your sales representative.
- You may also get an error informing you that the language of that document is unsupported. The GeoTagger does not support tagging in that language; that and other languages may be available in future releases.
- If you get one of these errors but believe your document's license or character set to be licensed or supported, you can force the GeoTagger to use your specified language or character set with the options on page 23.
- Some locations are in no country or province. When a location has no country or province record, the GeoTagger divides the world into a grid and tags each cell of the grid with the names of the countries and provinces that the cell covers. Some cells have no country or province; the GeoTagger leaves the relevant tag off of locations in such cells. When a cell covers multiple countries or provinces, the GeoTagger tags locations in such cells with lower country or province confidence.

Customer Support Contact Information

GSRP geOdrive customers should contact Schlumberger Customer Support. There are now three ways to contact Schlumberger Customer Support:

- Register in the portal at `http://support.slb.com` to submit requests for support
- E-mail ⟨`customercarecenter@slb.com`⟩
- Call customer support at +1 866 829 0234

All other customers should contact MetaCarta Customer Support:

- Toll-free: +1 888 458 0345, option #1
- International: +1 937 521 4200, option #1
- Email: ⟨`support@metacarta.com`⟩

Qbase business hours are 9:00am to 5:00pm Eastern Time, Monday through Friday, excluding company holidays. Calls outside of normal business hours will be forwarded to an answering service, and a member of our support team will return the call.