



METACARTA APPLIANCE  
GUIDE TO WEB SERVICES APIS  
Last modified on March 13, 2008  
Version 3.9

Copyright 2001 - 2008 MetaCarta, Incorporated. All rights reserved. Pat. 7,117,199. This product or document contains the confidential and proprietary information of MetaCarta, Incorporated ("MetaCarta"). This product or document is protected by copyright and distributed under license restricting its use, copying, distribution, disclosure and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of MetaCarta and its licensors, if any. Third-party software is copyrighted and licensed from MetaCarta suppliers.

Copyrighted and licensed third-party software used in this product includes: Debian Linux, which in turn includes the Open Source projects Tomcat Application Server and Perl and Python languages; software developed by Apache Software Foundation (<http://www.apache.org/>); DOM4J project from Metastuff Ltd.; and the Boost Regex Library from Dr. John Hammond (<http://www.boost.org>).

MetaCarta, the MetaCarta logo, CartaTrees, geOdrive, GeoTagger, GeoParser, and MetaCarta GTS are proprietary trademarks of MetaCarta that may be protected by one or more registrations. Other trademarks and service marks are the property of their respective owners.

All computer software provided by MetaCarta to the Government is Commercial Computer Software, as defined in Section 12.212 of the Federal Acquisition Regulation (48 CFR 12.212 (October 1995)) and Sections 227.7202-1 and 227.7202-3 of the Defense Federal Acquisition Regulation Supplement (48 CFR 227.7202-1, 227.7202-3 (June 1995)). Any Technical Data provided by MetaCarta shall be considered a "Commercial item" as defined in the FAR. The Government shall accept the standard commercial license for the commercial software and shall take no more than limited rights in any Technical Data describing any commercial item, component or process. In the event that, for any reason, the foregoing are deemed not applicable, the Government's right to use, duplicate or disclose any software shall be limited to "Restricted Rights" as defined in 48 CFR Section 52.227-19(c)(1) and (2) (June 1987), or DFARS 252.227-7014(a)(14) (June 1995), as applicable. Manufacturer is MetaCarta, Inc., 350 Massachusetts Avenue, 4th Floor, Cambridge, MA 02139

ALL PRODUCTS OR DOCUMENTATION ARE PROVIDED "AS IS" AND ALL EXPRESSED OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMER IS HELD TO BE LEGALLY INVALID.

## **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 Cambridge MA, Vienna VA, and Houston TX. To learn more about MetaCarta, please visit [www.metacarta.com](http://www.metacarta.com).

MetaCarta, Inc.  
350 Massachusetts Avenue, 4th Floor  
Cambridge MA 02139

Phone: + 1 617 661 6382  
Fax: + 1 617 661 6384  
Email: [info@metacarta.com](mailto:info@metacarta.com)

## Overview

The MetaCarta GTS appliance allows users to search indexed documents and stored geographic data based on both keywords and geographic references. The MetaCarta Web Services APIs allow you to perform searches programmatically, rather than using the provided graphical user interface. The JSON Search and KML Search APIs allow you to search for documents from your collection of ingested documents, while the JSON Location Finder API allows you to search for locations from Geographic Data Modules installed on your appliance.

The JSON Search API returns document search results in Javascript Object Notation (JSON) format, allowing you to integrate MetaCarta search results with your own systems however you would like. JSON is supported by most major languages — in many languages, integrating with JSON is much easier than integrating with SOAP. The JSON Search API results format makes it easy to create JavaScript-based graphical user interfaces based on MetaCarta's GeoSearch.js JavaScript library.

“AnyPage”, included in the Web Services Search APIs package, is a simple JavaScript application that demonstrates the use of the GeoSearch library. AnyPage allows you to add MetaCarta search results to any web page using just four lines of HTML (with some security limitations; see page 23). This application and its documentation are available at <http://metacarta.example.com/developers/AnyPage/> (where [metacarta.example.com](http://metacarta.example.com) is the hostname of your appliance).

The KML Search API uses a subset of the URL parameters available in the JSON Search API. The KML Search API provides KML 2.1 output specifically tailored for KML viewers such as ESRI ArcGIS Explorer, Google Earth, and NASA World Wind.

The JSON Location Finder API returns location search results in JSON format. The JSON formatted results are easy to integrate into customized applications, including graphical user interfaces, as described above. The JSON Location Finder API searches for locations in the Base MetaCarta GDM, as well as the IHS Global Oil and Gas GDM.

This document specifies the means for sending document or location queries to the GTS appliance, receiving the results of those queries, and parsing those results.

## Assumptions

While this document presents the Web Services APIs to the MetaCarta GTS appliance as simply as possible, a basic level of familiarity with programming is assumed. If you need additional assistance with the topics covered in this guide, please contact MetaCarta Support at [support@metacarta.com](mailto:support@metacarta.com).

The DNS name of your appliance is assumed throughout to be [metacarta.example.com](http://metacarta.example.com).

## Accessing the APIs

The Web Services APIs can be accessed through URLs on your appliance. Two URLs are available for each API: one for a secured version of each service, and one

for an unsecured version. The JSON Search API is accessible on your appliance at these two URLs:

<http://metacarta.example.com/services/search/JSON>  
<http://metacarta.example.com/services/search/JSON/secure>

The KML Search API is accessible on your appliance at:

<http://metacarta.example.com/services/search/KML>  
<http://metacarta.example.com/services/search/KML/secure>

The JSON Location Finder API is accessible on your appliance at:

<http://metacarta.example.com/services/location-finder/JSON>  
<http://metacarta.example.com/services/location-finder/JSON/secure>

The JSON Search API should be available on the GTS appliance; to confirm this, browse to <http://metacarta.example.com/developers/JSONExplorer/>. If you are prompted for a username and password, that means that your GTS administrator (possibly you) has configured security of some sort on the JSON Search API. See the “JSON Explorer” section on page 6 of this document for more information about the JSON Explorer. The JSON Explorer can also be used to determine access to the JSON Location Finder API.

If you believe you should have access to the JSON Search and JSON Location Finder APIs but get an error, the APIs might be disabled, or you might be using an old client developed with GET requests. For more information, please see page 23.

Similarly, to check the availability of the KML Search API, browse to <http://metacarta.example.com/clients/KMLSearch>. As in the case of the JSON Explorer, your appliance administrator may have configured security on the API.

If you get an unexpected error message when trying to access these URLs, please contact MetaCarta Support.

## Concepts

### JSON

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>.

### KML

Keyhole Markup Language is an XML-based markup language used to manage geographic data for graphical presentation. The KML Search API returns KML 2.1 files intended for use with KML viewers such as ESRI ArcGIS Explorer, Google Earth, and NASA World Wind. More information on KML is available at <http://code.google.com/apis/kml/documentation/index.html>.

### MetaCarta GTS Document Search Results

MetaCarta's default Web Search User Interface allows you to limit search results based on the following criteria:

- **Keyword search:** You can specify one or more keywords that must appear in any documents returned in your search results.
- **Geographic area:** You can specify a bounding box on a world map to limit your search results to those that contain references to that specific geographic area.
- **Time filter:** You can specify a temporal range to limit your search results to those that mention a time in that range.
- **Collection name:** You can specify a named collection to limit your search results to documents in that collection.

The JSON and KML Search APIs duplicate this search functionality. In addition, the APIs offer the following features:

- **Query extent gridding:** The geographic query extents can be subdivided into a grid of user-specified size. Separate searches are then performed in each cell of the grid.
- **Compressed output:** Large result sets can be transferred faster.
- **SRS:** The API provides reprojection and datum shifting to spatial reference systems of your choice.

Additionally, the user can specify a JavaScript handler function to the JSON Search API, which allows the JSON response to be loaded directly into an AJAX application.

### MetaCarta GDMs

A MetaCarta Geographic Data Module (GDM) includes a database of locations. The information for each location includes its place name, latitude and longitude, administrative district data, population, and type of place. A MetaCarta GTS appliance includes the Base GDM, and may include other specialized GDMs.

### MetaCarta Location Finder Results

The Location Finder included in MetaCarta's standard user interface allows you to search for locations based on place name and administrative districts. The JSON Location Finder API duplicates this functionality.

Location Finder results include locations from the Base GDM installed on your appliance, and, if installed, the IHS Global Oil and Gas GDM. The U.S. Street Address GDM and custom gazetteer, if any, are not covered by the Location Finder service.

## MetaCarta GTS Security

The MetaCarta Web Search Interface can accept both user authentication (proving that the user should be able to access the appliance) and user authorization (giving the user permission to view specific documents). The JSON and KML Search APIs each offer two URLs, one which requires authentication and one which does not. By visiting the URL that requires authentication, you can pass along both authentication (HTTP Basic Auth, Active Directory) information and authorization (Active Directory) credentials. For more information on how to do this, please see the Examples section on page 21. The JSON Location Finder API similarly has two URLs, one requiring authentication, and one that doesn't. In the case of the Location Finder service, the only concern is authentication for access to the service.

Using the JSON Search API or related tools, such as AnyPage, through HTTP GET in JavaScript has further security implications. Please see page 23 for a detailed discussion of the risks and techniques that can be used to minimize those risks.

**Note:** The KML Search API is not affected by the security concerns described on page 23.

## JSON Explorer

The JSON Explorer is a developer tool that allows authorized users to create and run JSON Search and JSON Location Finder searches and view the results in a browser window. The JSON Explorer is located at <http://metacarta.example.com/developers/JSONExplorer>. Access to the JSON Explorer is controlled using HTTP Basic Authentication (Basic Auth); if it is inaccessible, you may need to configure the permissions on your GTS appliance. For more information on configuring Basic Auth on your appliance, please see the “Basic Authentication” section of the *MetaCarta Appliance Administrator's Guide* or ask your appliance administrator for assistance. You can see the JSON explorer here:

The JSON Explorer allows you to change request parameters and see new results in realtime. The Query Parameters input boxes, a series of drop-down selection boxes and input text fields on the left side of the page, allow you to enter the same search parameters available in a standard JSON search request. The first drop down selection box allows you to choose between the Search API or Location Finder API. Only the input fields relevant to your selection will be available; other input fields will be greyed out.

When you have filled in all your desired request parameters, click the “Search” button. The large text box on the right side of the page will display the results of

the current search. Scroll bars at the right and bottom of the results allow you to see the entire results set.

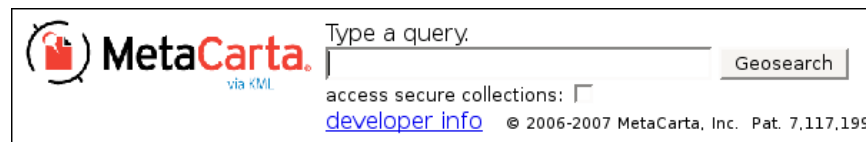
Above the results box are the Web Services Path display and the Parameters display box. The Web Services Path display shows the URL on the appliance where you would send an API request; the Parameters display shows the parameter string you would send to the API. The Parameters display automatically updates when you run a search by entering values into the Query Parameters inputs.

You can also change your query by editing the Parameters text box directly and then pressing enter.

When performing a document search with the JSON Explorer, the format of the output from each search is dependent on several parameters; the “action” field is particularly pertinent to the form of the output. For more information about the JSON Search API input parameters and output fields, see pages 10 and 14, respectively. For more information about the JSON Location Finder API input parameters and output fields, see pages 13 and 17.

## KML Search Client

The easiest way to interact with the KML interface is via the KML Search Client provided on your appliance at <http://metacarta.example.com/clients/KML>. This page allows you to enter a search request and receive results in different MIME types specified through the `format` parameter. For a description of the supported MIME types, see the input parameters section on page 10.



The KML Search Client is a static HTML page. You can copy that HTML and change its relatively referenced hyperlinks into absolute links. When a user enters search into the form field on that page, the browser loads a link into a hidden inline frame on the page. That link is to the KML Search API with `format=NetworkLink`, so the MIME type of the response causes the browser to load the contents with a KML viewer. If you have not selected a KML viewer to associate with that MIME type, the browser automatically asks you to choose an application. When the NetworkLink is loaded in the KML viewer, the viewer will load MetaCarta map labels for the user’s search using a `format=kmz` request to the KML Search API.

## Interface Definition

The JSON Search API allows you to make a variety of requests to a GTS appliance. Using the JSON Search API, you can search for documents, retrieve cached versions of these documents, and investigate the metadata attributes of documents and collections. Each different type of request is built using the corresponding *action* method. (For more information on the *action* parameter, see the Search Request Parameters section on page 10.) The requests take the form of a URL; the API returns results in JSON format.

The KML Search API allows you to search for documents with a URL request. The results from this API are returned in a format tailored for KML viewers.

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

The following sections describe the different types of requests for the APIs. The sections discuss how to construct the requests and how to interpret the returned results. Short examples are included in each section. An examples archive containing longer, more in-depth examples is available on the GTS appliance. See page 26 for information on how to extract the archive. Some language-specific examples are also described in the Examples section of this guide, starting on page 21.

## Searching

The JSON Search API and KML Search API both allow you to conduct document searches. These searches return a list of matching documents and geographic references, as well as information summarizing the search and its results.

### Location-Major and Document-Major Searches

The JSON Search API can return results ordered by location or by document. This preference is determined by the *action* parameter, which can be set to *getLocationReferences* or *getDocumentReferences*, respectively. In the location-major format, search results are indexed by location, and multiple documents referencing the same location are grouped together. In the document-major format, search results are indexed by document, and multiple locations referenced in the same document are grouped together. The same information is included in both formats. The location-major format is convenient for placing results on a map since they are grouped geographically, whereas the document-major format is more conducive to a straightforward list of documents.

The KML Search API only offers the *getLocationReferences* method.

### Major Search Parameters: Keyword, Time, and Space

Three major parameters help define searches sent through the JSON and KML Search APIs. The APIs support searches based on keyword, time range, and geographic area, as well as any combination of the three.

### Keyword Search

Keyword searching includes document keyword searching, site and url searching, and metadata attribute searching, including document collection specification.

To search for one or more document keywords, enter those keywords in the query string parameter 'query' as part of a search request URL. For example, in the line below, the search is for the keywords "Robert" and "Frost."

```
query=Robert Frost
```

The table below shows the formats for entering keywords and for entering some advanced search operators. Keyword searches are not case-sensitive.

| Operator                                   | Meaning   | Example                                 |
|--|---|---|
| <i>keyword</i>                             | results must contain <i>keyword</i>                       | Frost                                   |
| phrase in quotes                           | results must contain those words in that order            | "Robert Frost"                          |
| <i>-keyword</i>                            | exclude results that contain <i>keyword</i>               | -Frost                                  |
| <i>keyword</i>                             | increase relevance of results that contain <i>keyword</i> | Frost                                   |
| site: <i>domain keyword</i>                | search only the specified Web domain                      | site:example.com Frost                  |
| url: <i>url keyword</i>                    | search only the specified URL                             | url:http://example.com/index.html Frost |
| metadata:collection_name=(collection name) | search only the specified named collection                | metadata:collection_name=sample         |

### Geographic Area Search

The JSON and KML Search APIs also support searching by geographic area. You can create a latitude and longitude bounding box ('bbox' parameter) that will limit a query's geographic extent. By default, the geographic extent of a search is the whole earth. If you wish to specify a smaller geographic search area you must include both minimum and maximum latitudes and longitudes. Longitudes must fall in the range -180 to 180, latitudes, -90 to 90, and should be specified in degrees, including decimal degrees.

For example, the bounding box below would limit you search to documents containing references to locations in or near the State of Hawaii:

```
bbox=-160.6,18.8,-154.7,22.4
```

**Note:** The bounding box cannot wrap around the edges of the world projection, -180, -90, 180, 90. Minimum latitudes and longitudes must be smaller than maximum latitudes and longitudes.

### Time Search

Start and end time parameters allow you to filter by the time mentioned in the document. Searches default to an unlimited time range, but you may set a start time ('minextractedtime' parameter), end time ('maxextractedtime' parameter), or both. If you do so, the times must be integers representing hundredths of seconds since January 1, 1970, or "the epoch." (Various languages or software packages have conversion packages to do this for you; if you just want to look at one date and time, you can use an online conversion application like the one at <http://www.epochconverter.com/>). Dates and times before the epoch are represented by negative numbers, counting back from the epoch.

For example, the range below limits your search to documents referencing times between January 1, 1990, 00:00:00 GMT and January 1, 1991, 00:00:00 GMT:

```
minextractedtime=63115200000&maxextractedtime=66268800000
```

If you wanted to search for all documents referencing times since January 1, 1990 00:00:00 GMT with no upper bound, you could simply remove the 'maxextracted-time' parameter from the above example.

**Note:** GTS indexes dates in the range from 1902 to 2038. If a document contains dates outside of this range, they are simply ignored.

### Search URLs and Request Parameters

You can request information from the JSON and KML Search APIs by constructing a URL including any of the parameters specified in the search request parameters list below. A request URL has two parts. The first part is the search service's base URL on the appliance, 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 Search API URL containing the API version parameter and a query for the word "elephant" :

```
http://metacarta.example.com/services/search/JSON?version=1.1.0&query=elephant
```

Any parameter that you do not specify takes on its default value as given below. 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.

- **version** (Required)  
**Default value:** input other than the allowed value raises an error message  
**Allowed values:** 1.1.0  
**Usage:** The requestor must specify the API version they wish to use. This JSONExplorer sets the version=1.1.0 by default; the API requires a valid version parameter.
- **query** (Optional)  
**Default value:** (blank)  
**Allowed values:** string  
**Usage:** The search string to be sent to GTS. The query string can include operators such as 'site:domain.com' and collection name metadata.
- **bbox** (Optional)  
**Default value:** -180,-90,180,90  
**Allowed values:** a comma-separated string of minlon,minlat,maxlon,maxlat  
**Usage:** Defines the query's geographic extent.
- **srs** (Optional)  
**Default value:** epsg:4326  
**Allowed values:** any spatial reference system identifier supported by the GDAL library  
**Usage:** Causes the API to reproject and datum shift all output coordinates from unprojected WGS84 coordinates to the requested spatial reference system; also causes the API to expect the 'bbox' parameter to be specified in the spatial reference system requested by the 'srs' parameter. If not set, or if set to epsg:4326, no reprojection occurs. Some reprojections may slow search performance. See <http://www.gdal.org/gdal.utilities.html> for more.

- **action** (Required)  
**Default value:** getLocationReferences  
**Allowed values:** 'getLocationReferences', 'getDocumentReferences', 'getMetadataKeys', 'getMetadataValues', 'getCachedDocuments'  
**Usage:** Specifies the Search API function.
- **grid** (Optional)  
**Default value:** 1,1  
**Allowed values:** two integers separated by a comma  
**Usage:** The values used to make a grid out of the search area: the first is the number of cells horizontally and the second is the number vertically. If both numbers are 1, GTS will perform a single search. Otherwise, a separate search will be run on every grid cell, and these results will be merged into a single result set returned by the Search API. Using a grid larger than 1,1, while computationally intensive, helps ensure that the search results are more evenly dispersed across the entire searched area.
- **startref** (Optional)  
**Default value:** 0  
**Allowed values:** positive integers  
**Usage:** Specifies the first result to return; startref=0 corresponds to the first reference provided by the database for each grid cell in the query. See the 'grid' parameter for more details. This parameter is useful for creating an interface that lets users page through results.
- **maxrefs** (Required)  
**Default value:** 40  
**Allowed values:** positive integers  
**Usage:** Limits the number of georeferences returned for each grid cell in the query. See the 'grid' parameter for more details. This parameter is useful for creating an interface that lets users page through results.
- **minextractedtime** (Optional)  
**Default value:** no limit  
**Allowed values:** integer  
**Usage:** Limits returned georeferences to have extracted times later than 'minextractedtime'.
- **maxextractedtime** (Optional)  
**Default value:** no limit  
**Allowed values:** integer  
**Usage:** Limits returned georeferences to have extracted times earlier than 'maxextractedtime'.
- **key** (Optional)  
**Default value:** (blank)  
**Allowed values:** string  
**Usage:** A key='MetadataKey string' parameters must be included in a request for getMetadataValues. To get values for more than one key, include additional key='string' phrases.
- **doctoken** (Optional)  
**Default value:** empty list  
**Allowed values:** comma-separated list of strings  
**Usage:** One or more docid='DocID string' parameters must be included in a request for getDocumentContents.
- **densitygrid** (Optional)  
**Default value:** None  
**Allowed values:** two integers separated by a comma  
**Usage:** Setting density grid will return document density data for the given search, divided up into as many cells as there are in the densitygrid. The first integer is the number of cells horizontally and the second is the number vertically. Setting densitygrid requires the setting of densityformat. 60,40 is the recommended maximum, as it is computationally expensive to calculate the density output.

- `densityformat` (Optional)

**Default value:** None

**Allowed values:** `Reference, Matrix`

**Usage:** Setting `densityformat` is required when `densitygrid` is set. The density data will be returned as either a reference (to be used with other services, like the density web service), or an array of values indicating the density for each populated cell in the `densitygrid`.

- `callback` (Optional)

**Default value:** None

**Allowed values:** `string`

**Usage:** The `'callback'` parameter is only used when doing requests via GET. (Using `'callback'` with POST will return an unreadable response.) This parameter facilitates the use of the `'script-adding'` technique to load data from the GTS, 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 GTS's server response is simply the calling of the specified callback function, with the results object as the first parameter.

## Using the JSON Location Finder

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.

### Major Location Finder Parameter: Place Name

Location 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 and press enter or click on the "submit" button. 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 administrative 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 clients, including the JSON Explorer, also 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 21) does URL-encode the query, so %2C is sufficient.

### Location Finder URLs and Request Parameters

You can request information from the JSON Location Finder API by constructing a URL including any of the parameters specified in the search request parameters below. A request URL has two parts. The first part is the search service's base URL on the appliance, 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 locations named "Elephant" :

```
http://metacarta.example.com/services/  
location-finder/JSON?version=1.1.0&query=elephant
```

Any parameter that you do not specify takes on its default value as given below. As with the Search API, the service responds to erroneous input values with an error message.

- **version** (Required)  
**Default value:** input other than the allowed value raises an error message  
**Allowed values:** 1.1.0  
**Usage:** The requestor must specify the API version they wish to use. The JSON Explorer sets the version=1.1.0 by default; the API requires a valid version parameter.
- **query** (Required)  
**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."
- **geometry** (Optional)  
**Default value:** None  
**Allowed values:** WKT  
**Usage:** Setting geometry for a location finder query will return the geometry for any locations for which it is available.
- **callback** (Optional)  
**Default value:** None  
**Allowed values:** string  
**Usage:** The 'callback' parameter is only used when doing requests via GET (using 'callback' with POST will return an unreadable response). This parameter facilitates the use of the 'script-adding' technique to load data from the GTS, 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 GTS's server response is simply the calling of the specified callback function, with the results object as the first parameter.

### How to Read Search 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.

KML uses tags to structure and define elements. Typically, tags come in open-close pairs. These two tags surround contents. Tags that define objects may include the XML identifier of that object. For example, an object might look like:

```
<object id="xml-id">
  <attribute>contents</attribute>
</object>
```

For more information about KML output, please see the KML 2.1 Reference along with the other documentation at <http://code.google.com/apis/kml/documentation/>

## JSON Search Result Fields

When you send a `action=getLocationReferences` or `action=getDocumentReferences` request, the appliance will return a JSON object containing the results of your search. The output of a successful search contains three general categories of parameters: input query parameters, system parameters, and result references and other results parameters.

Several of the input query parameters are repeated as *key:value* pairs. These confirm that the appliance acted on your requested parameters and illustrate the various default values implicitly set in your request. These parameters are shown below:

- `MinExtractedTime` is an integer showing the minimum time you requested, or a blank string if you set no minimum time filter.
- `MaxExtractedTime` is an integer showing the maximum time you requested, or a blank string if you set no maximum time filter.
- `StartRef` is an integer showing the first reference you requested.
- `MaxRefs` is an integer showing the maximum number of references that you requested.
- `Query` is a string showing the query you sent to the appliance.
- `BBox` is a dictionary of values containing the bounding box you entered. It contains 'X' and 'Y' as well as latitude and longitude. If you entered an 'srs' parameter, 'X' and 'Y' will show the coordinates you requested, and the latitude and longitude will show the translated values that the appliance used to conduct your search.
- `SRS` is a string showing the SRS that you requested.

- `Grid` is a comma separated list containing the two integers you requested as a grid.
- `UserCredentials` is an object containing information about the user credentials used to perform the search. The object contains a `Username`. If no username is supplied with the request, the value of `Username` returned will be null.

The JSON object contains several output fields that describe the system and its corpus of documents.

- `SystemVersion` is the GTS version and API version of the appliance that you are communicating with.
- `NumDocsCorpus` is an integer representing the number of searchable documents in the entire corpus available to this user.
- `NumRefsCorpus` is an integer representing the number of document references occurring in the entire corpus available to this user.
- `Styles` is a dictionary containing graphical representation information for each result style.

Typically, the bulk of the JSON object is the result set and related information. In addition to the result documents or locations list, there are output parameters that provide an overview of the search results, including density results.

- `Warnings` is a list of warnings (as strings) resulting from your search.
- `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 result set.
- `ApproxNumDocsQuery` is an approximation of the number of documents with references matching your query with the correct number of significant digits.
- `ApproxNumRefsQuery` is an approximation of the number of document references that match your query with the correct number of significant digits.
- `Density` is an object containing density results:
  - `Matrix` is a density matrix in array form. Each item in the array represents an entry in a sparse matrix. These entries are also in array form, containing two subarrays, as shown below:

```
[ [[a1,b1],[c1,d1,e1]], [[a2,b2],[c2,d2,e2]], ... ]
```

The first subarray represents the location of the entry in the matrix. The first entry in the subarray is the coordinate of the density cell in the longitudinal grid, with a coordinate of 0 representing the lowest longitude segment of the grid defined by the 'bbox' and 'densitygrid' input parameters. The second entry is the coordinate in the latitudinal grid, with a coordinate of 0 representing the lowest latitude segment of the grid.

The second subarray represents density data from the query for that grid cell. The first entry in the subarray is the *reference count*, the approximate number of document references that match your query in that grid cell. The second entry is the *summed geoconfidence* for all the references matching your query in the cell, and the third is the *summed query relevance* for all the references.

**Note:** Grid cells with no query results will not be represented in the density matrix. Reference count, summed geoconfidence, and summed query relevance for these cells should be assumed to be 0.

- `Reference` is a string giving a reference to density information, usable with other services.
- `Documents` or `Locations` is a list of Document or Location objects containing the following components:
  - `References` is a list of references to the Location or from the Document.
  - `HighestRelevance` is a floating point number indicating the highest relevance reference within this Document or Location object's list of References.
  - `DocToken` is a non-human-readable string that uniquely identifies the document in the database from which this reference came. You need this string to obtain the cached document (if cached documents are enabled on your appliance). There is a DocToken in each Document object in the document-major format and in each Reference object in the location-major format. This DocToken can be used with the query action 'getDocumentContents' to retrieve the cached document.
  - `Title` is the title string extracted from the document. There is a Title in each Document object in the document-major format and in each Reference object in the location-major format.
  - `URL` is the Universal Resource Locator associated with the document. There is a URL in each Document object in the document-major format and in each Reference object in the location-major format.
  - `ExtractedTime` is an object with two parameters: 'Extract', the time string extracted from the document, and 'Time', the seconds since the UNIX epoch that the appliance believes the extracted time represents. All the references from a given Document have the same Time, so there is one Time in each Document object in the document-major format and one Time in each Reference object in the location-major format.
  - `LocID` is a non-human-readable string that uniquely identifies a tagged location. There is one LocID in each Location object in the location-major format and in each Reference object in the document-major format.
  - `Georef` is a human-readable string indicating the name of the tagged location. It is in the Reference object in both formats.
  - `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. It is in the Location objects in the location-major format and in the Reference objects in the document-major format.
  - `Centroid` is an object with four elements: 'Latitude' and 'Longitude', which are floating point numbers indicating a center point of the location in unprojected coordinates, and 'X' and 'Y', which are coordinates projected into the SRS provided as part of the request. The centroid is in the Location objects in the location-major format and in the Reference objects in the document-major format.
  - `GeoConfidence` is a floating point number indicating the probability that the author of the reference intended to refer to this location, as calculated by MetaCarta GTS. It is in Reference objects in both formats.

- `GeoExtract` is the string that the appliance extracted from the document and considers geographic. The `GeoExtract` contains HTML formatting that highlights the portions of the text that indicate the location. You may want to use this text and the `Extract` text to create map labels. It is in the `Reference` objects in both formats.
- `Relevance` is a floating point number indicating the importance of this reference to the total query, that is, to the free-text query and also to the geographic location selected by the map extent. It is in the `Reference` objects in both formats.
- `Extract` is a string extracted from the document that contains text relevant to the query string. This is in the `Reference` objects in both formats.

A simple way to become familiar with the format and contents of search results is to use the JSON Explorer, as described on page 6. Sample results are available in the Examples Archive on the appliance. (To extract and access this archive, see page 26.)

The mechanics of obtaining these types of output are discussed in language-specific sections in the Examples section this document (see page 21); you can also see an example search results package in the Examples archive.

If you run the example JSON search client included in the search examples archive, you can work with these data structures easily. After importing the `GeoSearch` class, you can parse both the output of a `getDocumentReferences` request and then the output of a `getLocationReferences` request.

### JSON Location Finder Result Fields

When you send a Location Finder request, the appliance 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 appliance.
- `SystemVersion` is the GTS version and API version of the appliance with which you are communicating.
- `Styles` is a dictionary containing graphical representation information for each result style.

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.
- `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 containing information for the location. One value it will contain is '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, visit <http://www.census.gov/geo/www/fips/fips.html>.
  - `Type` 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.
  - `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.

JSON Explorer output provides a means to become familiar with JSON Location Finder API results. A sample JSON Explorer Location Finder output is included in the Examples Archive. See page 26 for instructions on extracting and accessing the archive.

See the Examples section this document (starting on page 21) for language-specific examples on producing results.

## Formatting and Compressing Results

The JSON Search API and JSON Location Finder API use HTTP as their 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 'Accept-Encoding' HTTP Request Header that contains 'gzip', the JSON APIs 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 halve the time that a client must wait to retrieve content. Many web browsers set this header automatically, helping performance in AJAX applications.

The KML Search API also supports output compression through the use of the `format` parameter:

- Setting `format=kmz` causes the API to return KMZ compressed output with its `Content-type` header set to `application/vnd.google-earth.kmz`.
- Setting `format=kml` causes the API to return KML compressed output with its `Content-type` header set to `application/vnd.google-earth.kml+xml`.
- Setting `format=NetworkLink` causes the API to return KML compressed output with its `Content-type` header set to `application/vnd.google-earth.kml+xml`.

Any other value for `format` including empty retrieves uncompressed XML that most browsers display directly.

## Additional JSON Search API Actions

There are several other useful operations that can be performed by the JSON Search API. The JSON Search API allows you to retrieve the full text of a document from the appliance. You can also investigate the existing metadata fields and retrieve their values. These operations can be called by setting the `action` parameter in a request to `getDocumentContents`, `getMetadataKeys`, or `getMetadatKeyValues`, respectively. These actions are not supported for the KML Search API.

### Getting a Whole Document

Once you have performed a search, you may want to get the entire text of a document from the appliance. The appliance stores converted text copies of all documents, not the original documents themselves; for the original document, you should just go to the URL returned by the appliance for that document. However, sometimes the original document is inaccessible, or you want just the plaintext version. In these cases, you can send the JSON Search API the `action=getDocumentContents` request along with the `DocToken` of each document you wish to retrieve.

You can try this in the JSON Explorer, or use the following curl request:

```
curl -d
"version=1.0.0&action=getDocumentContents&doctoken=(SEE
BELOW) "
http://metacarta.example.com/services/search/JSON
```

The `DocToken` string is typically long, e.g.:

```
MDAwNTczZjllOGFhY2RhNzU5MjlkMzYyMGIxYWEyZDQsbG9jYWxob3N
0OjE2MDM5XzExNjQ4MzU2NjBfMTE2NDgzNTczM18wMDoxMzo3Mjo2
Nzo1MToxZiwwZiI4==
```

The response to such a request is:

```
{
  "DocumentContents" : [
    {
      "DocToken" : "MDAwODIzNWF1MzY3MDA1YWMxYTMxNjcwZDA1Ym
        YwMDgsbG9jYWxob3N0OjE2MDE1XzExOTU3MTg2
        MDVfMTE5NTcxOTAwOF8wMDoxMT00MzpkMjo4MD
        pkOCwyNzclMw==",
      "Content-Type" : "text/plain",
      "Data" : "FULL TEXT OF DOCUMENT",
      "Character-Encoding" : "ascii"
    }
  ],
  "SystemVersion" : "MetaCarta GTS v3.9.0, JSON search API v1.1.0",
  "ResultsCreationTime" : "Thu Nov 29 19:40:39 2007 UTC"
}
```

If you request a non-existent document, you will get a blank response.

For more information on using curl, see page 22.

### Getting a List of Metadata Keys

To get a list of potential metadata keys, simply send the appliance a `getMetadataKeys` request:

```
curl -d "version=1.0.0&action=getMetadataKeys"
      http://metacarta.example.com/services/search/JSON
```

The `MetadataKeys` return field is a list of metadata keys that can be passed into the API via the 'key' parameter or used in the 'query' string. Typically, one of the `MetadataKeys` is `collection_name`. Named collections allow appliance administrators to create logical groups of searchable documents.

A typical response to a `action=getMetadataKeys` request is:

```
{
  "MetadataKeys" : [
    "collection_name",
    "extractor_stage"
  ],
  "SystemVersion" : "MetaCarta GTS v3.9.0, JSON search API v1.1.0",
  "ResultsCreationTime" : "Thu Nov 29 19:55:34 2007 UTC"
}
```

### Getting Metadata Key Values

To get the potential values of one or more metadata keys, send the appliance a `getMetadataValues` request with key values for the keys you are interested in, as follows:

```
curl -d "version=1.0.0&action=getMetadataValues&key=
collection_name&key=another_parameter"
http://metacarta.example.com/services/search/JSON
```

The `MetadataValues` return field is a dictionary of lists. Each requested 'key' parameter string becomes the name of an attribute of the object, and each key's value is a list of the possible values for the metadata key. If you request values for a non-existent key, you will get a null response for that key.

The URL request above would generate an output like this:

```
{
  "SystemVersion" : "MetaCarta GTS v3.9.0, JSON search API v1.1.0",
  "ResultsCreationTime" : "Thu Nov 29 20:17:25 2007 UTC",
  "MetadataValues" : {
    "collection_name" : [
      "col-0",
      "col-1",
      "col-2",
      "col-3",
      "col-4"
    ]
  }
}
```

## Examples

### GTS 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`. For more information on creating a basic auth user for use with the Ingestion API, see the "Basic Authentication" section of the *MetaCarta Appliance Administrator's Guide*.

### Security

To pass security credentials on to the Appliance, you must use a different URL; for example, for the JSON Search API, instead of <http://gts.example.com/services/search/JSON>, you must make requests to <http://gts.example.com/services/search/JSON/secure>. Security credentials must be passed on through middleware; while many middleware providers support HTTP Basic Auth, very few support AD. Passing Active Directory credentials through the HTTP transport layer requires SPNEGO (the Simple and Protected GSS-API NEGOTiation mechanism; for more information on SPNEGO, please see <http://en.wikipedia.org/wiki/SPNEGO>).

**Note:** In some cases, passing AD security credentials to the JSON API through JavaScript may be impossible; please see page 23.

On Linux or similar operating systems, we recommend recent versions of libcurl. Windows provides two DLLs that offer this functionality: WININET.DLL and WINHTTP.DLL. We recommend WINHTTP.DLL. It is also possible to implement an entirely new middleware, though doing so requires significantly more programming.

None of the examples that ship with this manual are configured to use Active Directory authentication; if you need help developing with Active Directory, please contact MetaCarta Support at [support@metacarta.com](mailto:support@metacarta.com).

In addition to document security, there are some security risks involved with instantiating objects from the JSON returned as search results. In some languages, it is possible to simply call

```
myObject = eval(json formatted string)
```

but it is possible for an attacker to insert a function into the string and cause your `eval()` statement to run code you did not mean to execute. Therefore, it is better to parse JSON output using a JSON parser. JSON parsers are available for most languages at <http://www.json.org>. Using a JSON parser like Python's *simplejson* allows you to safely instantiate objects from JSON formatted strings:

```
import simplejson
myObject = simplejson.loads(json formatted string)
```

There are also known language-specific security risks. Consult the appropriate section before using or creating a search client in a given language.

## 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 fetch URLs, you can use it to send search requests. For example, to make a request to the JSON Search API of the appliance [gts.example.com](http://gts.example.com), run the following command:

```
host:~$ curl -d "version=1.1.0"
http://fred:ginger@gts.example.com/services/search/JSON
```

A number of example requests are included on your appliance in the archive `/usr/share/doc/metacarta/WebServicesExamples.tgz`. Instructions for accessing these files are on page 26.

## Python

Python has an HTTP library included in the base distribution and you can download *simplejson* from most repositories of Python packages. <http://cheeseshop.python.org/pypi/simplejson/1.4> is a link to the *simplejson* package in one such repository.

The sample Python output below will search the appliance [gts.example.com](http://gts.example.com) for the string "oil" and return the results in location-major format. The code used can be accessed on the appliance; please see page 26 for instructions.

```
>>> from JSONclient import GeoSearch
>>> search = GeoSearch({'version': '1.1.0', 'q': 'oil'})
>>> results = search.getLocationReferences()
>>> results
<JSONclient.Results instance at 0xb7d9bb2c>
>>> results.NumDocsCorpus
48424
...
```

The Examples archive also includes two code samples showing how to use the JSON Search API and JSON Location Finder API:

- `JSONSearchClient.py`: A simple Python client that can send search requests to the JSON Search API and receive results.
- `JSONLocationFinderClient.py`: A simple Python client that can send search results to the JSON Location Finder API and receive results.

If you need a Python reference, the freely available book *Dive Into Python* by Mark Pilgrim is an excellent reference for reasonably experienced programmers. You can access it at <http://diveintopython.org>. Other resources are also available at the Python project's official website, <http://www.python.org>.

## JavaScript

There are two files included in the Examples archive demonstrating how to use the JSON Search API in Javascript:

- `JSONSearchClient.js`: A simple Javascript client that can send search requests to the JSON Search API and receive results.
- `JSONHtml.html`: An HTML file with embedded calls to `JSONSearchClient`.

One of the benefits of using the JSON APIs is the ability to use the APIs to request data from different hosts in the browser without any server-side dependency. You do this via the addition of 'script' tags to the body of the page, which can load remote content and return the data to a callback function.

### Risk for Attack

Unfortunately, using the JSON APIs in JavaScript — and thus using the GeoSearch and AnyPage functionality it makes possible — is a major security risk. 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 GTS appliance is available to all users within a private network. The information indexed by the GTS 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 GTS appliance 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 GTS appliance contains secured content which is only accessible to individuals with appropriate security credentials. When a trusted user makes HTTP GET requests to the JSON API, 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 GTS appliance 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 JSON API 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 appliance ships with GET requests to the JSON APIs turned off entirely. See page 26 in the following section for a description of how to change this setting.

## Solutions

For most applications, you can simply use HTTP POST instead of HTTP GET to send requests to the JSON APIs. However, browsers use GET requests to retrieve files specified in SCRIPT elements. Browsers also allow POST requests via the XMLHttpRequest object, but by default only permit POST requests to be sent to the originating host. This restriction means that POST cannot be used to retrieve information from other hosts. Because of this restriction, retrieving JavaScript files from hosts other than the originating host requires the use of GET. Therefore, this security hole cannot be closed simply by switching to POST.

However, it is possible to send POST requests to a proxy server on the same host as the JSON API client that can then relay those requests back to the MetaCarta appliance. There are many ways to set up such a proxy server, including configuring web server software to proxy automatically and running a CGI-based proxy on the server sending JSON API requests. Setting up a proxy is best handled by a systems administrator in your organization; we have included here an Apache 1.3 stanza that would configure a proxy on your local webserver to gts.example.com:

```
# Load the proxy module for Apache 1.3
LoadModule proxy_module /usr/lib/apache/1.3/libproxy.so

# Limit all requests to the service to HTTP POST
<Location /search-proxy>
  <LimitExcept POST>
    Deny from all
  </LimitExcept>
</Location>

# Disable proxying in general, and disabling caching for the service
# URL, then forward requests for the service to the appliance.
<IfModule mod_proxy.c>
  ProxyRequests Off
  NoCache /search-proxy
  ProxyPass /search-proxy http://gts.example.com/services/search/JSON
  ProxyPassReverse /search-proxy http://gts.example.com/services/search/JSON
</IfModule>
```

If you are using Apache 2.0, simply change the first few lines to read:

```
# Load the proxy modules for Apache 2.0
LoadModule proxy_module /usr/lib/apache2/modules/mod_proxy.so
LoadModule proxy_http_module /usr/lib/apache2/modules/mod_proxy_http.so
```

If you configure a proxy, you do not need to enable GET requests. However, in some low-risk circumstances, you may choose to enable GET requests on either your insecure JSON API service or your secure JSON API service. First, to see the status of your JSON API services, run this command:

```
metacarta:~$ sudo webservices_json_control status
```

**Note:** Using the `webservices_json_control` command requires either `sudo` or root access.

If you have not changed your configuration, both JSON and secure JSON will be turned off, as follows:

| Service               | Risk     |
|-----------------------|----------|
| -----                 | ----     |
| /location-finder/JSON | Low Risk |
| /search/JSON          | Low Risk |
| /search/JSON/secure   | Low Risk |

You can then enable GET requests with the following command:

```
metacarta:~$ sudo webservices_json_control
  enable-insecure-get /search/JSON
```

Afterwards, your status output should look like this:

| Service               | Risk                      |
|-----------------------|---------------------------|
| -----                 | -----                     |
| /location-finder/JSON | Low Risk                  |
| /search/JSON          | GET enabled, risk of data |
| /search/JSON/secure   | Low Risk                  |

If you want to disable GET requests, just use the `disable-insecure-get` option to `webservices_json_control`.

**Note:** GET requests are limited in length by the client browser; due to limitations in Internet Explorer, we recommend not using requests longer than 1,500 characters. Longer requests may yield incorrect or inconsistent results.

## Java

Java's `java.net.URLConnection` class can be used to talk to the JSON APIs. You may wish to use one of the JSON parsers listed at <http://json.org/>.

Please contact MetaCarta Support at [support@metacarta.com](mailto:support@metacarta.com) for more help with the JSON Search API using this language.

## C/C++

Using C/C++, you have access to the library version of `curl` (see page 22). You may wish to use JSON parsers, such as `jsoncpp` and `zoolib`, listed at <http://www.json.org>.

Please contact MetaCarta Support at [support@metacarta.com](mailto:support@metacarta.com) for more help with the Web Services Search API using this language.

## C#/VB.NET (.NET)

The .NET platform provides the `System.Net.HttpWebRequest` class, which can be used to communicate with the JSON APIs. You may wish to use one of the JSON parsers listed at <http://json.org/>.

Please contact MetaCarta Support at [support@metacarta.com](mailto:support@metacarta.com) for more help with the Web Services Search APIs using this language.

## Extracting the Examples Archive

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

```
metacarta:~$ tar -xvzf
  /usr/share/doc/metacarta/WebServicesExamples.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 on the appliance. To do this, run

```
metacarta:~$ cd /usr/share/doc/metacarta
```

before running the command above.