



METACARTA APPLIANCE  
GUIDE TO WEB SERVICES APIS  
Last modified on April 27, 2009  
Version 4.1

Copyright ©2001 - 2009 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.; the PocketSOAP library (<http://www.pocketsoap.com>); 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 area. 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. The JSON Location Finder API allows you to search for locations from Geographic Data Modules installed on your appliance. The JSON Query Parser API separates unstructured queries into keywords and place names so that they can be processed using the Search and Location Finder APIs or other tools.

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. The JSON Search API results format makes it easy to create JavaScript-based graphical user interfaces and is encapsulated in the MetaCarta JavaScript SDK.

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, IHS Global Oil and Gas GDM, and Street Address GDMs.

The JSON Query Parser API creates interpretations of unstructured queries. The API breaks an input query into geographic and non-geographic parts, and returns a query with the non-geographic keywords and a list of the most likely intended geographic locations. (The use of the JSON Query Parser API requires a licensed copy of the GeoTagger.) The query is returned as a JSON object.

The JSON Saved Search and Notification API allows you to interface with the Saved Search and Notification service. Using the JSON Saved Search and Notification API, you can create, edit, delete, and view saved searches in the system.

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

The 2.0.0 version of these APIs are described in this document. The 1.1.0 version of the Search and Location Finder APIs is deprecated; for stand-alone documentation of that version, please see past documentation.

## **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 Customer Support (see page 42).

The DNS name of your appliance is assumed throughout to be `metacarta.example.com`.

## Accessing the APIs

You can access the Web Services APIs through URLs on your appliance. For the Search and Saved Search and Notification APIs, two URLs are available for each API: one for a secured version of each service, and one for an unsecured version. The Location Finder and Query Parser APIs are only available through unsecured URLs.

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>

The JSON Query Parser API is accessible on your appliance at:

<http://metacarta.example.com/services/query-parser/JSON>

The JSON SSN API is accessible on your appliance at:

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

These APIs should all be available; to confirm this, browse to <http://metacarta.example.com/developers/JSONExplorer/>. See the “JSON Explorer” section on page 7 of this document for more information about the JSON Explorer.

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 APIs. If you believe you should have access to the JSON 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 37.

Similarly, to check the availability of the KML Search API, browse to <http://metacarta.example.com/clients/KMLSearch>. Again, 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 Customer Support (see page 42).

## 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 and Street Address GDMs. The custom gazetteer, if any, is not searchable through the Location Finder service.

### Unstructured Queries and the Query Parser

MetaCarta's document and location searching tools, including the MetaCarta Web User Interface, Search APIs, and Location Finder API, are designed to be used with specific kinds of query input. For example, if you wanted to search for documents about houses in Maine using the Web Search Interface, you would use the Location Finder to select the searchable region for the State of Maine, and search using the keyword "houses". This would provide more accurate and complete results than performing a keyword search on the phrase "houses in Maine".

The Query Parser API is designed to take an unstructured query string, such as "houses in Maine", and divide the query into geographic and non-geographic parts suitable for use with MetaCarta's other search tools.

### Saved Search and Notification

The Saved Search and Notification (SSN) system allows end users to save searches, including keywords, named collections, time filtering, and geographic extent limits, on the appliance and access them again at a later date through user interfaces or through automatic notifications. The SSN API allows you to interface with the SSN Manager, the software on the appliance that manages this functionality for end users.

### 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 Search and Saved Search and Notification APIs 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 35. The JSON Location Finder API and JSON Query Parser API each have only one URL that does not require authentication.

Using the JSON Search API or related tools through HTTP GET in JavaScript, including through the MetaCarta JavaScript SDK, has further security implications. Please see page 37 for a detailed discussion of the risks and techniques that can be used to minimize those risks.

**Note:** The KML APIs are not affected by the security concerns described on page 37.

## JSON Explorer

The JSON Explorer is a developer tool that allows authorized users to create and run JSON API requests 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:

MetaCarta Web Services

Alter query parameters to update the JSON results. [detailed documentation](#)

Query Parameters

API: Search

action: getLocationReferences

query:

bbox: -180,-90,180,90

Web Service Path: /services/search/JSON

Parameters:

version=2.0.0&action=getLocationReferences&maxrefs=40&grid=1,1&bbox=-180,-90,180,90&srs:

JSON results:

```
{
  "MaxExtractedTime" : ""
}
```

[KML](#)

The JSON Explorer allows you to change request parameters and see new results in real time. 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 or other API request. The first drop down selection box allows you to choose between the Search API, Location Finder API, Saved Search API, and Query Parser API. The second allows you to select the *action* parameter you wish to use with that API. The appropriate parameter fields for your selected API and action will be displayed.

When you have filled in all your desired request parameters, click the “Submit Query” 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.

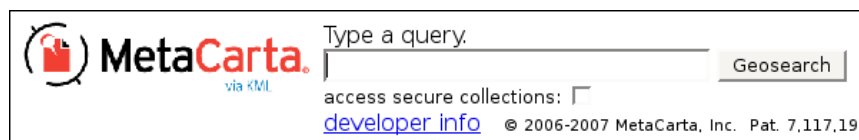
You can also change your query by editing the Parameters text box directly and then pressing enter.

When performing any request 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 11 and 15, respectively. For more information about the JSON Location Finder API input parameters and output fields, see pages 22 and 23. For more information about the JSON Query Parser

API input parameters and output fields, see pages 25 and 26. For more information about the JSON Saved Search and Notification API input parameters and output fields, see pages 29 and 34, respectively.

## 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 11.



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, your browser should automatically ask 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 those documents, and investigate the metadata attributes of documents and collections. Each 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 11.) 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 JSON Query Parser API allows you to send query strings to the appliance to be parsed. The parsing request takes the form of a URL; results are returned in JSON format.

The JSON Saved Search and Notification API allows you to interact with the SSN manager to add, edit, or delete saved searches as well as view the saved searches in the system.

The following sections describe the different types of requests for the APIs and discuss how to construct the requests and how to interpret the returned results.

Short examples are included in each section. An examples archive containing in-depth examples is available on the GTS appliance; see page 40 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 35.

## API Requests

You can request information from the JSON and KML APIs 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 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=2.0.0&query=elephant>

As discussed on page 4, all services have unsecured base URLs that can be used to construct these requests; some services also have secured base URLs. Individual sections for each API discuss the parameters used with that service.

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.

## Formatting and Compressing Results

The JSON Search, Location Finder, Query Parser, and Saved Search and Notification APIs 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.

## API Output

The JSON APIs return JSON objects, while the KML APIs return KML output. One way to become familiar with the format and contents of search results is to use the JSON Explorer, as described on page 7. Sample results are available in the Examples Archive on the appliance. (To extract and access this archive, see page 40.) Similarly, the KML Search Client, discussed on page 8, is a good tool for exploring KML Search results.

The mechanics of obtaining these types of output are discussed in language-specific sections in the Examples section this document (see page 35); you can also see an example search results package in the Examples archive.

### 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.

### How to Read KML Results

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/>

## The JSON and KML Search APIs

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.

### Search URLs and Requests

The base URLs for the JSON and KML Search service are given on page 4 and an explanation of constructing requests is given on page 9.

The following sections describe the parameters used in creating different types of search requests.

### The action Parameter

The *action* parameter determines the function that the Search API performs. The default action is a document reference search, but the Search API can also return the content of a document, a list of the metadata keys in the system, or the values of a specific metadata key.

- *action* (Required)  
**Default value:** `getLocationReferences`  
**Allowed values:** `getLocationReferences`, `getDocumentReferences`, `getMetadataKeys`, `getMetadataValues`, `getDocumentContents`  
**Usage:** Specifies the Search API function.

### Search actions: `getLocationReferences` and `getDocumentReferences`

The actions *getLocationReferences* and *getDocumentReferences* both perform searches through the JSON Search API. The parameter determines if the API returns results ordered by location or by document. 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 the output 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` action.

### 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 searching by collection name.

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. Multiple operators and keywords can be combined in a space-separated list. Keyword searches are not case-sensitive.

Operator	Meaning	Example
<i>keyword</i>	results must contain <i>keyword</i>	Frost
<i>keyword keyword2</i>	results must contain all keywords	Frost poetry
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</i>	search only the specified Web domain	site:example.com
site: <i>domain keyword</i>	search only the specified Web domain for <i>keyword</i>	site:example.com Frost
url: <i>url keyword</i>	search only the specified URL	url:http://example.com/index.html
metadata:collection_name= <i>(collection name)</i>	search only the specified named collection	metadata:collection_name=sample
feature:fips_id= <i>FIPS ID</i>	search only the specified administrative region (for more information, see the <i>MetaCarta GeoTagger API Guide</i> )	feature:fips_id=MA25
feature:type= <i>feature type</i>	search only for results associated with the specified feature type (for more information, see the <i>MetaCarta GeoTagger API Guide</i> )	feature:type=SCH

## Geographic Area Search

The JSON and KML Search APIs also support searching by geographic area. You can create a latitude and longitude bounding box using the *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 your 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 or the service will return a warning.

### 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 using the *minextractedtime* parameter, set an end time using the *maxextractedtime* parameter, or both. If you do so, the times must be integers representing hundredths of seconds since January 1, 1970, “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 ‘maxextractedtime’ 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.

### getLocationReferences and getDocumentReferences Parameters

The following parameters are used with a *getLocationReferences* or *getDocumentReferences* Search request.

- **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 in the coordinate system defined by ‘srs’.
- **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](http://www.gdal.org/gdal_utilities.html) for more.

- **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'.
- **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 JSONExplorer sets the version by default, the API requires a valid version parameter.
- **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 the 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.
- **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:** 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.

### **getLocationReferences and getDocumentReferences Result Fields**

When you send a JSON Search API `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.
- `Errors` is a list of errors (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 an appropriate 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 (`[a1, b1]`) represents the location of the entry in the matrix. The first entry in the subarray (`a1`) 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 (`b1`) is the coordinate in the latitudinal grid, with a coordinate of 0 representing the lowest latitude segment of the grid.

The second subarray (`[c1, d1, e1]`) represents density data from the query for that grid cell. The first entry (`c1`) in the subarray is the *reference count*, the approximate number of document references that match your query in that grid cell. The second entry (`d1`) is the *summed geo-confidence* for all the references matching your query in the cell, and the third (`e1`) 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 taken from the document 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 suggested for the display of this reference in a map. The 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 API results is to use the JSON Explorer, as described on page 7. JSON results are shown on the main JSON Explorer page, while the “KML” link at the right of the page provides a direct link to the KML Search *getLocationReferences* results. Additionally, sample results are available in the Examples Archive on the appliance. (To extract and access this archive, see page 40.) The mechanics of obtaining these types of output are discussed in language-specific sections in the Examples section this document (see page 35).

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.

### Search action: *getDocumentContents*

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. The parameters that can be used with this action are shown below:

- *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 JSONExplorer sets the version by default, the API requires a valid version parameter.
- *doctoken* (Optional)  
**Default value:** empty list  
**Allowed values:** comma-separated list of strings  
**Usage:** One or more *doctoken='DocID string'* parameters must be included in a request for *getDocumentContents*.
- *callback* (Optional)  
**Default value:** None  
**Allowed values:** string  
**Usage:** 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.

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

```
curl -d
"version=2.0.0&action=getDocumentContents&doctoken=(SEE
BELOW) "
http://metacarta.example.com/services/search/JSON
```

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

```
MDAwNTczZjllOGFhY2RhNzU5MjlkMzYyMGIXYWEyZDQsbG9jYWxob3N0
00je2MDM5XzExNjQ4MzU2NjBfMTE2NDgzNTczM18wMDoxMzo3Mjo2
Nz0lMT0xZiZwMzI4==
```

**Note:** For more information on using curl, see page 36.

### getDocumentContents Results

The response to a request created using the `getDocumentContents` action is:

```
{
  "Errors" : [
  ],
  "Warnings" : [
  ],
  "DocumentContents" : [
    {
      "DocToken" : "MDAwODIzNWFlMzY3MDA1YWMxYTM5NjcwZDA1Ym
        YwMDgsbG9jYWxob3N00je2MDE1XzExOTU3MTg2
        MDVfMTE5NTcxOTAwOF8wMDoxMT00MzpkMjo4MD
        pkOCwyNzclMw==",
      "Content-Type" : "text/plain",
      "Data" : "FULL TEXT OF DOCUMENT",
      "Character-Encoding" : "ascii"
    }
  ],
  "SystemVersion" : "MetaCarta GTS v4.0.0, JSON search API v2.0.0",
  "ResultsCreationTime" : "Mon Jun 02 15:33:40 2008 UTC"
}
```

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

### Search action: getMetadataKeys

To get a list of potential metadata keys using the JSON Search API, send the appliance a `getMetadataKeys` request:

```
curl -d "version=2.0.0&action=getMetadataKeys"
http://metacarta.example.com/services/search/JSON
```

The following parameters can be used with such a request:

- **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 JSONExplorer sets the version by default, the API requires a valid version parameter.

- `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.
- `callback` (Optional)  
**Default value:** None  
**Allowed values:** string  
**Usage:** 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.

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.

### getMetadataKeys Results

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

```
{
  "Errors" : [
  ],
  "Warnings" : [
  ],
  "MetadataKeys" : [
    "collection_name",
    "sample_key"
  ],
  "SystemVersion" : "MetaCarta GTS v4.0.0, JSON search API v2.0.0",
  "ResultsCreationTime" : "Mon Jun 02 15:29:01 2008 UTC"
}
```

### Search action: getMetadataValues

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

```
curl -d "version=2.0.0&action=getMetadataValues&key=
collection_name&key=another_parameter"
http://metacarta.example.com/services/search/JSON
```

The following parameters can be used with such a request:

- `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 JSONExplorer sets the version by default, the API requires a valid version parameter.

- `callback` (Optional)

**Default value:** None

**Allowed values:** string

**Usage:** 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.

### getMetadataValues Results

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:

```
{
  "Errors" : [
  ],
  "Warnings" : [
  ],
  "SystemVersion" : "MetaCarta GTS v4.0.0, JSON search API v2.0.0",
  "ResultsCreationTime" : "Mon Jun 02 15:36:10 2008 UTC",
  "MetadataValues" : {
    "collection_name" : [
      "Sample1",
      "Sample2",
      "Sample3"
    ]
  }
}
```

## 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. The base service URLs are shown on page 4, and an explanation of constructing requests is given on page 9. Here is an example JSON Location Finder API URL request that queries for locations named "Elephant" :

<http://metacarta.example.com/services/location-finder/JSON?version=2.0.0&query=elephant>

## 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 Parameter: 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.

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 35) does URL-encode the query, so %2C is sufficient.

### 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)  
**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 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 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.

### 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.
- `RegionReference` is a hashed reference to region data for your query that is stored on the appliance. This output parameter should be ignored; it is currently only used internally.
- `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.
- `Errors` is a list of errors (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 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, 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 40 for instructions on extracting and accessing the archive.

See the Examples section this document (starting on page 35) for language-specific examples on producing results.

## The JSON Query Parser API

The JSON Query Parser allows you to parse unstructured queries into query strings usable with other services. Each query request returns the most likely interpretation of the query, including a non-geographic keyword phrase and the highest confidence geographic name along with the most probable locations associated with that name.

## Query Parser URLs and Request Parameters

You can request a parsed query from the Query Parser API by constructing a URL including any of the parameters specified in the Query Parser parameter list below. The base service URLs are shown on page 4, and an explanation of constructing requests is given on page 9. Here is an example JSON Query Parser API URL containing the API version parameter, the geometry parameter set to false, and a query string “houses in Maine” :

<http://metacarta.example.com/services/query-parser/JSON?version=2.0.0&geometry=false&query=houses in Maine>

## Query Parser Parameters

The following sections describe the parameters that can be used with the JSON Query Parser API. The Query Parser has one characteristic parameter.

### Major Query Parser Parameter: Query String

The query string is the primary input parameter for the Query Parser. The query string should include one or more words intended for use as search terms, separated by spaces. While there is no theoretical limit to the length of the query string, limits on URL length and on the input file size to the MetaCarta GeoTagger impose practical limits, and very long queries may not be effective searches when sent to a MetaCarta Search API.

### Query Parser Parameter List

The following parameters can be used with the Query Parser service:

- `version` (Required)  
**Default value:** None  
**Allowed values:** 2.0.0  
**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)  
**Default value:** (blank)  
**Allowed values:** string  
**Usage:** The query string is the input for which a Query Parser search is initiated.
- `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. Results outside this extent are ignored. If this is not specified, the bounding box contains the whole globe.
- `geometry` (Optional)  
**Default value:** None  
**Allowed values:** WKT  
**Usage:** Setting the geometry format for a query parser request will return the geometry in that format for any locations for which it is available.

- `minqueryconfidence` (Optional)  
**Default value:** 0  
**Allowed values:** floating-point number greater than or equal to 0 and less than or equal to 1  
**Usage:** If a query has a confidence lower than *minqueryconfidence*, it will not be returned. Note that this may be limited by `maxqueries`.
- `minlocationweight` (Optional)  
**Default value:** 0  
**Allowed values:** floating-point number greater than or equal to 0 and less than or equal to 1  
**Usage:** No locations with a weight lower than *minlocationweight* will be returned. Note that this may be limited by `maxlocations`.
- `maxqueries` (Optional)  
**Default value:** None  
**Allowed values:** integers greater than or equal to 1  
**Usage:** The maximum number of queries to return. Note that the number of queries returned may also be limited by `minqueryconfidence`. (2.0.0 version will return only one query.)
- `maxlocations` (Optional)  
**Default value:** None  
**Allowed values:** integers greater than or equal to 1  
**Usage:** The maximum number of locations to return per query. Note that the number of locations returned may also be limited by `minlocationweight`. (2.0.0 version will return only one location.)

### JSON Query Parser Result Fields

When you send a Query Parser request, the appliance will return a JSON object containing the results of your query parsing.

Just as in a document search or location finder request, the output of a successful query parser request contains three general categories of results: input query parameters, system parameters, and result locations and other results parameters. Input query parameters confirm that the appliance acted on your requested parameters and illustrate the various default values implicitly set in your request.

- `Query` is a string showing the query you sent to the appliance.
- `MinQueryConfidence` is the *minqueryconfidence* sent to the appliance. If a value was not specified in the request, `MinQueryConfidence` will be zero.
- `MinLocationWeight` is the *minlocationweight* sent to the appliance. If a value was not specified in the request, `MinLocationWeight` will be zero.
- `BBox` is a dictionary of values containing the bounding box you entered. It contains latitude and longitude.

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 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 query listings. Each result query contains data about the relevance of the query, the non-geographic part of the query, and the geographic part of the query, including extensive information about possible

locations corresponding to the query, similar to output from the Location Finder API.

- `Warnings` is a list of warnings (as strings) resulting from your search.
- `Errors` is a list of errors (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.
- `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'.
- `ViewBox` is a dictionary of values containing the minimum and maximum latitude and longitude of a minimal bounding box containing all the locations returned in *Queries*.
- `Queries` is a list of query interpretations as parsed by the Query Parser API. Each query returned is an object containing the following fields:
  - `Confidence` is the probability as calculated by the Query Parser that this is the correct interpretation of the query string in the request.
  - `RemainingQuery` is the non-geographic portion of the query for this interpretation of the input query.
  - `GeoRef` is the tagged geographic word or phrase considered in this interpretation of the input query.
  - `Locations` is a list of Location objects corresponding to the *GeoRef* of this query, containing the following components:
    - \* `Weight` is the probability that the *GeoRef* intends to refer to this location.
    - \* `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. This output is only returned when the *geometry* parameter inputs a valid geometry type.
    - \* `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 an 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 Query Parser API results. A sample JSON Explorer Query Parser output is included in the Examples Archive. See page 40 for instructions on extracting and accessing the archive.

See the Examples section this document (starting on page 35) for language-specific examples on producing results.

## The JSON Saved Search and Notification API

The JSON Saved Search and Notification (SSN) API allows you to make a variety of requests to a GTS appliance. Using the JSON SSN API, you can add new saved searches, edit existing searches, execute searches, and view the existing searches and their status. Each different request is built using the corresponding *action* method. The requests take the form of a URL; the API returns results in JSON object format.

Unlike the other APIs discussed in this guide, the JSON SSN API allows you to alter part of the GTS user interface. Requests sent through this API can add, delete, or change saved searches in the SSN manager. The use of this API should be restricted to system administrators or regulated through user interfaces.

### Saved Search and Notification URLs and Request Parameters

You can send a request to the JSON Saved Search and Notification API by constructing a URL including any of the parameters specified in the request parameters lists in the following sections. The base service URLs are shown on page 4, and an explanation of constructing requests is given on page 9. Here is a JSON SSN API URL containing the API version parameter, the action parameter *listSearches* and the sort-by key *owner*:

```
http://metacarta.example.com/services/saved-search/JSON?version=2.0.0&action=listSearches&sortBy=owner
```

### Saved Search and Notification Parameters

The following sections describe the parameters that can be used with the JSON Saved Search and Notification API. The Saved Search and Notification has one characteristic parameter.

### Major Saved Search and Notification Parameter: *action*

The *action* parameter, shown below, determines what function the request you send to the JSON Saved Search and Notification API will perform. Based on the action parameter that you send the JSON SSN API, your request can add a new saved search, edit an existing search, execute an existing search, allow you to view an existing search or the status of the Saved Search and Notification system.

- *action* (Required)  
**Default value:** None  
**Allowed values:** `addSearch`, `deleteSearch`, `renameSearch`, `updateSearchSchedule`, `addNotifyees`, `deleteNotifyees`, `listSearches`, `execute`, `status`,  
**Usage:** Specifies the SSN API function.

The functionality of each action and the additional parameters required or optional for that action are specified in the following sections.

One additional parameter is required in all cases. The *version* parameter specifies what version of the JSON SSN API to use with the request. Currently, only one version of the API is available.

- *version* (Required)  
**Default value:** None  
**Allowed values:** `2.0.0`  
**Usage:** The requestor must specify the API version they wish to use. Although the JSONExplorer sets the version by default, the API requires a valid version parameter.

### JSON SSN API Action: *addSearch*

The action *addSearch* allows you to create a new search in the Saved Search and Notification system. Additional parameters allow you to specify the properties of the search, including the name of the saved search, the keyword query string, geographic filter, and time filter, and the list of Saved Search and Notification users who will be able to access the search or receive automatic notifications from the search, along with their notification information. (For more information on creating and modifying Saved Search and Notification Users, please see the *MetaCarta Appliance Administrator's Guide*.) The following parameters can be used with the *addSearch* action:

- *name* (Required)  
**Default value:** None  
**Allowed values:** the name of a search  
**Usage:** Specifies the search to be created or modified with this action.
- *query* (Optional)  
**Default value:** (blank)  
**Allowed values:** string  
**Usage:** Specifies the keyword query string to be used by the saved search.
- *bbox* (Optional)  
**Default value:** `-180,-90,180,90`  
**Allowed values:** a comma-separated string of `minlon`, `minlat`, `maxlon`, `maxlat`  
**Usage:** Defines the saved search's geographic extent. Results outside this extent are ignored. If this is not specified, the bounding box is the whole globe.

- `queryminextractedtime` (Optional)  
**Default value:** None  
**Allowed values:** integer  
**Usage:** Limits georeferences returned by the search to have extracted dates and times later than 'queryminextractedtime'. The time and date are in hundredths of a second since epoch time.
- `querymaxextractedtime` (Optional)  
**Default value:** None  
**Allowed values:** integer  
**Usage:** Limits georeferences returned by the search to have extracted dates and times earlier than 'querymaxextractedtime'.
- `executionfrequency` (Optional)  
**Default value:** 60 seconds  
**Allowed values:** integer  
**Usage:** Sets the frequency in seconds with which a search is re-executed. This number is rounded to the next highest multiple of 60 seconds.
- `numnotifyees` (Optional)  
**Default value:** 0  
**Allowed values:** non-negative integers  
**Usage:** The size of the group of notifyees to be altered with this command.
- `notifyeetype` (Required)  
**Default value:** None  
**Allowed values:** SMTP, XMPP, RSS, NNTP  
**Usage:** This specifies the type of notification you wish to send to the user with the corresponding index. Each field should be indexed, from 0 to `numnotifyees-1`. For example, the first user should be designated by the expression `notifyeetype_0=<type>`. The available types for each search are shown in the output of a `listSearches` request, and may include Jabber (XMPP), email (SMTP), RSS feed (RSS), and Netnews (NNTP).
- `notifyeeaddress` (Optional)  
**Default value:** None  
**Allowed values:** string  
**Usage:** This is the address corresponding to the type of notification sent to the user with the same index. (See `notifyeetype` for a description of indexing.) For Jabber, this is the screen name of the user. For email, this should be the email address of the user. For an RSS feed, this should be the URL of the feed. For Netnews, this is be the name of the news group to which notifications should be posted.
- `uiversion` (Required)  
**Default value:** None  
**Allowed values:** atlantis, classic  
**Usage:** Specifies the version of the user interface that will used in the creation of notification results links for this saved search.
- `browserstate` (Optional)  
**Default value:** None  
**Allowed values:** string  
**Usage:** A free-form string representing the browser state for client use.

See the example for the action `addNotifyee` on page 32 for more information on specifying notification recipients.

#### JSON SSN API Action: `deleteSearch`

The action `deleteSearch` allows you to delete an existing search from the Saved Search and Notification system. The following parameter must be used with this

action:

- **name** (Required)  
**Default value:** None  
**Allowed values:** the name of a search  
**Usage:** Specifies the search to be created or modified with this action.

#### JSON SSN API Action: *renameSearch*

The action *renameSearch* allows you to change the name of an existing saved search. The following parameters must be used with this action:

- **name** (Required)  
**Default value:** None  
**Allowed values:** the name of a search  
**Usage:** Specifies the search to be created or modified with this action.
- **newname** (Required)  
**Default value:** None  
**Allowed values:** string  
**Usage:** Specifies the new name that you wish to assign to the search.

#### JSON SSN API Action: *updateSearchSchedule*

The action *updateSearchSchedule* allows you to change the execution frequency for an existing saved search. The following parameters must be used with this action:

- **name** (Required)  
**Default value:** None  
**Allowed values:** the name of a search  
**Usage:** Specifies the search to be created or modified with this action.
- **executionfrequency** (Optional)  
**Default value:** 60 seconds  
**Allowed values:** integer  
**Usage:** Sets the frequency in seconds with which a search is re-executed. This number is rounded to the next highest multiple of 60 seconds.

#### JSON SSN API Action: *updateSearchRealTime*

The action *updateSearchRealTime* allows you to change the search execution frequency used for real time browser notification for an existing saved search. (A newly-created saved search does not produce real time notification by default.) The following parameters must be used with this action:

- **name** (Required)  
**Default value:** None  
**Allowed values:** the name of a search  
**Usage:** Specifies the search to be created or modified with this action.

- `realtimefrequency` (Required)

**Default value:** None

**Allowed values:** integer

**Usage:** The execution frequency used for real time browser notification, in seconds. This number will be rounded to the next highest multiple of 60 seconds.

#### JSON SSN API Action: `addNotifyees`

The action `addNotifyees` allows you to add notifyees to an existing search. The notification type and address of each notifyee should also be set with the command. The following parameters should be used with this action.

- `numnotifyees` (Optional)

**Default value:** 0

**Allowed values:** non-negative integers

**Usage:** The size of the group of notifyees to be altered with this command.

- `notifyeetype` (Required)

**Default value:** None

**Allowed values:** SMTP, XMPP, RSS, NNTP

**Usage:** This specifies the type of notification you wish to send to the user with the corresponding index. Each field should be indexed, from 0 to `numnotifyees-1`. For example, the first user should be designated by the expression `notifyeetype_0=<type>`. The available types for each search are shown in the output of a `listSearches` request, and may include Jabber (XMPP), email (SMTP), RSS feed (RSS), and Netnews (NNTP).

- `notifyeeaddress` (Optional)

**Default value:** None

**Allowed values:** string

**Usage:** This is the address corresponding to the type of notification sent to the user with the same index. (See `notifyeetype` for a description of indexing.) For Jabber, this is the screen name of the user. For email, this should be the email address of the user. For an RSS feed, this should be the URL of the feed. For Netnews, this is be the name of the news group to which notifications should be posted.

For example, if you wished to add an email notification recipient with the email address `(user@example.com)` to the pre-existing search “ExampleSearch”, you could send the following request to the appliance:

```
http://metacarta.example.com/services/saved-search/JSON?version=2.0.0&action=
addNotifyees&name=ExampleSearch&numnotifyees=1&notifyeetype_0=
SMTP&notifyeeaddress_0=user@example.com
```

It is recommended that you check the notification types enabled for each search using the `listSearches` action. You will be able to submit name and address pairs for all notification types, but only pairs of the types listed in the “AvailableNotifyeeTypes” output parameter will receive notifications. For more information about output parameters, see page 34.

#### JSON SSN API Action: `deleteNotifyees`

The action `deleteNotifyees` allows you to delete notifyees from an existing search. The following parameters should be used with this action:

- `numnotifyees` (Optional)  
**Default value:** 0  
**Allowed values:** non-negative integers  
**Usage:** The size of the group of notifyees to be altered with this command.
- `notifyeeid` (Required)  
**Default value:** None  
**Allowed values:** Notifyee ID  
**Usage:** The notifyee ID of the notification type/address pair whose status you wish to alter for this search. (Notifyee IDs are listed by the `listSearches` commance.) Each field should be indexed, from 0 to `numnotifyees-1`. For example, the first notification type/address pair should be designated by the expression `notifyeeid_0=UID`.

To find the `notifyeeID` for a notification recipient, use the `listSearches` action. The notifyee type/notifyee address pairs for each search are numbered starting at 1. For example, to remove two notifyees, "1" and "12", from the search "Example-Search" you could send the following request to the appliance:

```
http://metacarta.example.com/services/saved-search/JSON?version=2.0.0&action=deleteNotifyees&name=ExampleSearch&numnotifyees=2&notifyeeid_0=1&notifyeeid_1=12
```

#### JSON SSN API Action: `listSearches`

The action `listSearches` returns a list of all the stored searches in the system, sorted by the key specified in the parameter `sortby`. The following parameters should be used with this action:

- `sortby` (Optional)  
**Default value:** none  
**Allowed values:** `id, owner, searchname, searchstring, minlatitude, maxlatitude, minlongitude, maxlongitude, timebetweenruns, timecreated, timelastrun, timelastnotified, latestdoctime`  
**Usage:** The name of the column to use as the sort key.
- `sortorder` (Optional)  
**Default value:** `ASC`  
**Allowed values:** `ASC, DESC`  
**Usage:** The sort order to use with the given `sortby` key, ascending (`ASC`) or descending (`DESC`).

#### JSON SSN API Action: `execute`

The action `execute` allows you to execute an existing stored search. This action can be used on saved searches whether or not they have an assigned execution frequency. The following parameter should be used with this action:

- `name` (Required)  
**Default value:** None  
**Allowed values:** the name of a search  
**Usage:** Specifies the search to be created or modified with this action.

### JSON SSN API Action: status

The action *status* returns the status of the Saved Search and Notification system. No additional parameters are used with this action.

### JSON SSN API Result Fields

When you send a request URL, the appliance will return a JSON object containing the results of your request.

For the action parameters *addSearch*, *deleteSearch*, *renameSearch*, *updateSearch-Schedule*, *updateRealTimeSchedule*, *addNotifyees*, *deleteNotifyees*, *execute*, and *status*, the results JSON object will contain only status information. The returned information includes:

- `Warnings` is a list of warnings (as strings) resulting from your request.
- `Errors` is a list of error codes resulting from your request. These error codes each represent a specific type of error.
- `SystemVersion` is a human-readable string specifying the GTS version and SSN API version used for this saved search.
- `NotifyeeTypes` is a dictionary of notification protocol types and their descriptions.
- `Searches` is a blank list, used as a placeholder.

The output from a request with a *listSearches* action includes all of the above fields, but the *Searches* output field contains a list of searches:

- `Searches` is a list of the saved searches on the system. Each search is an object containing the following parameters:
  - `Owner` is the username of the Saved Search and Notification user that owns the search. In a system that does not require authorization, the username may be null. The format of the username is dependent on the authorization type used.
  - `Name` is the name of the search.
  - `ID` is the identification number for this saved search.
  - `BrowserState` is a free-form string representing the browser state input during the creation of this search.
  - `QueryParameters` is an object containing the parameters of the search. These parameters include:
    - \* `Query` is the query string for this search.
    - \* `MinExtractedTime` is the minimum time and date for this search, presented in hundredths of a second since epoch.
    - \* `MaxExtractedTime` is the maximum time and date for this search, presented in hundredths of a second since epoch.
    - \* `BBox` is an object representing the geographic bounding box for this query. It contains four fields representing the sides of the bounding box:
      - `MinLatitude` is the minimum latitude.
      - `MaxLatitude` is the maximum latitude.

- `MinLongitude` is the minimum longitude.
- `MinLatitude` is the maximum longitude.
- `ExecutionFrequency` is the execution frequency of the search, in seconds. If the execution frequency is 0, then the search will not be automatically executed.
- `RealTimeFrequency` is the execution frequency used for real time browser notifications, in seconds. If the real time execution frequency is 0, then the search will not be automatically executed.
- `CreationTime` is the creation time and date for this search in hundredths of a second since epoch.
- `LastExecuteTime` is the last time that the search was executed in hundredths of a second since epoch.
- `LastNotificationTime` is the last time a notification was sent for this search in hundredths of a second since epoch.
- `LatestDocTime` shows the ingestion time of the latest document found with this search in hundredths of a second since epoch.
- `Notifyees` is a list containing the notifyees of this search. Each notifyee is represented as an object containing the following fields:
  - \* `Id` is the notifyee identification number.
  - \* `Transport` is the notification mechanism used for this notifyee.
  - \* `Address` is the notifyee's address for the chosen transport.
- `AvailableNotifyeeTypes` is a list containing the notification types available for this saved search.

A simple way to become familiar with the format and contents of search results is to use the JSON Explorer, as described on page 7. A sample `listSearches` result is available in the Examples Archive on the appliance. (To extract and access this archive, see page 40.)

## 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 APIs through JavaScript may be impossible; please see page 37.

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 Customer Support (see page 42).

In addition to document security, there are some security risks involved with instantiating objects from the JSON returned as 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=2.0.0"
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/WebServicesSearchExamples.tgz`. Instructions for accessing these files are on page 40.

## 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 Examples archive 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 location finder requests to the JSON Location Finder API and receive results.

Example Python clients for the JSON Query Parser API and JSON Saved Search and Notification API are also available on the appliance:

- `/usr/lib/metacarta/JSONQueryParserClient.py`: A simple Python client that can send query parser requests to the JSON Query Parser API and receive results.
- `/usr/lib/metacarta/JSONSavedSearchClient.py`: A simple Python client that can send query parser requests to the JSON Saved Search and Notification 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

The MetaCarta JavaScript SDK allows you to design your own web applications using MetaCarta functionality or add MetaCarta functionality to existing web pages and web applications. For more information, see the *MetaCarta JavaScript SDK Guide*.

While the JavaScript SDK provides search functionality suitable for most applications, you can also create your own JavaScript clients to interface with the Web Services APIs. Whether using the SDK or these APIs, you can request data from different hosts in the browser without any server-side dependency. To do this, add 'script' tags to the body of the page, which can load remote content and return the data to a callback function.

If you are using the JavaScript SDK or creating your own JavaScript applications to interface with the APIs, please heed the following security concerns.

### Risk for Attack

Unfortunately, using the JSON APIs in JavaScript can be 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 appliance, 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 appliance 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 39 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 services or your secure JSON API services. First, to see the status of your JSON API services, run this command:

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

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
/saved-search/JSON/secure	Low Risk
/search/JSON/secure	Low Risk
/saved-search/JSON	Low Risk
/query-parser/JSON	Low Risk

You can then enable GET requests with the following command:

```
metacarta:~$ 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 siphoning
/saved-search/JSON/secure	Low Risk
/search/JSON/secure	Low Risk
/saved-search/JSON	Low Risk
/query-parser/JSON	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 Customer Support (see page 42) for more help with the JSON Web Services APIs using this language.

## C/C++

Using C/C++, you have access to the library version of `curl` (see page 36). You may wish to use JSON parsers, such as `jsoncpp` and `zoolib`, listed at <http://www.json.org>.

Please contact Customer Support (see page 42) for more help with the JSON Web Services APIs 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 Customer Support (see page 42) for more help with the JSON Web Services 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/WebServicesSearchExamples.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.

## **Troubleshooting**

If your older applications developed for MetaCarta Web Services APIs no longer work properly, and you are receiving errors like the following: "All MetaCarta APIs require a "version" parameter as input in the URL," you need to change your applications to use a more recent version of the API specification. Only the two most recent versions of the API are supported; for the JSON Search API, for example, that would be 2.0.0 and 1.1.0.

In 4.0.0, the JSON and KML Search APIs only allow you to search for 10,000 documents at a time. If you receive an error like "Requested 30000 results; searching for more than 10000 results is not supported," you should use `minResult` and `maxResult` to select the first 10000 results or any other selection of results you want to see.

## Customer Support Contact Information

GTS 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](mailto:customercarecenter@slb.com)
- Call customer support at +1 866 829 0234

All other customers should contact MetaCarta Customer Support:

- Toll-free: +1 866 661 6382, option #3
- International: +1 617 661 6382, option #3
- Email: [support@metacarta.com](mailto:support@metacarta.com)

MetaCarta 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 MetaCarta employee will return the call.