Term Lookup API

Brought to you by IData, Inc

Table of contents

1 Introduction

The term lookup service lets you lookup specific terms using either the precise term name (ignoring case), the term ID, or the ID of a specific version of a term. If no match is found, the response will return a status of 0, but there will not be any results in the body of the request.

2 Request Specification

The DNS name in the URL to which you submit a request takes care of telling us the institution whose terms you want to lookup. Requests will be submitted over the Internet via the HTTP protocol. Our whole application is accessed securely via HTTPS, so while our framework supports both secure and non-secure requests, this API will only accept HTTPS connections. Some Data Cookbook API requests may be asynchronous. The term lookup API is not. Term matches will be returned in the HTTP response to the lookup request.

2.1 URL

This service is provided at:
`https://<your_subdomain>.datacookbook.com/institution/terms/lookup`

2.2 Parameters

Authentication

- **pw** – password of the user who will be authenticating the request. This could be encoded or encrypted. Encoding would be straightforward. Encrypting becomes more complicated because of managing certificates and differences in encoding implementations on different platforms and systems, and between different programming languages.
- **un** – username of user who will be authenticating the request.
- **OR** – include authentication token(s) retrieved using the service_login service as cookies in the request.

Required

- **lookup** – string you want to search for. Could be a specific term name, or could be the ID of a term, or of a specific term version. The matchType parameter must match the value you pass in lookup. So, if you pass a name, it defaults to "exact_text", so you don't have to set matchType. If you pass an ID, though, matchType needs to reflect the type of ID you are looking up.
- **requestType** – type of request. For term lookup, the request type is "term_lookup".

Optional

- **jsonFunction** – Name of function you want JSON to be passed to on return.
- **jsonVariable** – Name of variable you want JSON to be assigned to on return.
- **matchType** – type of matching you want to use to select results that are returned. Supported values are "term_id", "version_id", or "exact_text". Defaults to "exact_text". A "term_id" lookup looks for a term with the id specified in the "lookup" parameter. A "version_id" lookup looks for a term with a version that has the ID specified in the "lookup" parameter. An "exact_text" lookup just looks for the exact term name you specify as the full name of a term in our system, ignoring case.
- **outputFormat** – Format you want the results to be rendered in. Potential supported formats: "rss", "html" (basic, with simple classes on each element), "xml", or "json". Default is "xml".

2.3 Format

You can choose from one of two request formats: form input parameters or request XML.

2.3.1 HTTP Query String Request Format

If you choose to implement requests using form input parameters, a search request would be either:

- An HTTP(S) GET request that has each of the required parameters above in its query string, and that could also contain any of the other parameters.
- An HTTP(S) POST request that has each of the required parameters above stored as form inputs in the body of the request, and that could also contain any of the other parameters.

You should only place each parameter in the request once. If you accidentally place a parameter there twice and the two instances have two different values, we can't guarantee which will be used in processing your request.

Sample HTTP GET query string request:

```
https://idata.datacookbook.com/institution/terms/lookup?un=jonathanmor
gan&pw=nopeeking!&requestType=term_lookup&lookup=hungry&matchType=exac
t_text&outputFormat=rss
```

2.3.2 XML Request Format

The XML request contains the same information, but it is more structured, and is in an XML transaction dialect that is used by all of our XML-based APIs. The XML request in our dialect contains a list of request parameters and <Parameter> elements stored in a <ParameterList>. The required and optional parameters differ for each request type.

Parameters are stored in a ParameterList element, one to a Parameter element. Each parameter contains a <Name> and <Value> element. This structure could accommodate much more complicated nested parameter structures if needed (nested ParameterLists inside a <Value> element, etc.), but for now, we are just planning on implementing name-value pairs.

An XML request is sent as the body of an HTTP(S) request to the service you want to invoke.

Sample XML service transaction request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceTransaction xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
    <ServiceRequest serviceName="term_lookup">
        <ParameterList>
            <Parameter>
                <Name>un</Name>
                <Value>jonathanmorgan</Value>
            </Parameter>
            <Parameter>
                <Name>pw</Name>
                <Value>nopeeking!</Value>
            </Parameter>
            <Parameter>
                <Name>requestType</Name>
                <Value>term_lookup</Value>
            </Parameter>
            <Parameter>
                <Name>lookup</Name>
                <Value>hungry</Value>
            </Parameter>
            <Parameter>
                <Name>matchType</Name>
                <Value>exact_text</Value>
            </Parameter>
            <Parameter>
                <Name>outputFormat</Name>
                <Value>rss</Value>
            </Parameter>
        </ParameterList>
    </ServiceRequest>
</ServiceTransaction>
```

3 Response Specification

The lookup results will be the body of the HTTP(S) response to the HTTP(S) request for term lookup. The response can be returned in a number of formats: RSS, HTML, XML, and JSON.

If you plan on accessing services using AJAX, then JSON or HTML will make more sense, though either RSS or XML would be implementable, as well.

If you are planning on interacting with this programmatically on a server, then it probably makes more sense to use either RSS or XML, though some server-side languages let you parse JSON, too, and can also do a passable job treating HTML like XML since we return xhtml.

Fields presented in all response formats :

- **term name** – Name of the term.
- **functional definition** – text explanation of the term. In the system, there are two definitions possible for each term, a functional and a technical definition. The functional definition is the text explanation of the term, in as close to plain English as possible. The technical definition is intended to contain specifics on retrieving a given term out of a database, or technical directions about interpreting it or displaying it. We don't return this information for search results.
- **term URL** – This will be the URL of the data cookbook page where a user can go to see more data on the term (like the technical definitions, associated terms, tags applied to the term, where it fits in the hierarchy of terms, and the functional areas to which it belongs).

3.1 RSS Response Format

In an RSS response, an RSS document is the body of the HTTP(S) response, with each item containing a term that matches the submitted lookup criteria. For now, the lookup implementation will either return one term if there is a match, or no terms if not. No terms are returned if there is an error.

For each item returned, the term name is the <title>, the term's functional description is the <description>, the last time that term was updated is stored in the <pubDate> element, and the URL to the Data Cookbook page for the term is in the <link> and <guid> elements.

Our API implements RSS 2.0.

Sample RSS service transaction response:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rss version="2.0">
    <channel>
        <title>IData Data Cookbook term_lookup results</title>
        <description>Here's what we found for your lookup on
"hungry".</description>

 <link>http://idata.datacookbook.com/institutions/terms/search.xml</li
nk>
        <language>en-us</language>
        <copyright>Copyright 2010, IData, Inc.</copyright>
        <generator>IData Inc. Data Cookbook</generator>
        <managingEditor>bparish@idatainc.com (Brian Parish, President
and CEO, IData, Inc.)</managingEditor>
        <webMaster>kdezio@idatainc.com (Ken Dezio, CTO, IData,
Inc.)</webMaster>
        <docs>http://blogs.law.harvard.edu/tech/rss</docs>
        <pubDate>Fri, 05 Feb 2010 05:00:00 CST</pubDate>
        <lastBuildDate>Fri, 05 Feb 2010 13:30:08 EST</lastBuildDate>
        <ttl>30</ttl>
        <item>
            <title>hungry</title>

 <link>http://idata.datacookbook.com/institution/terms/12345</link>
            <description><![CDATA[A hungry student is one who has a
need that he or she feels must be met.  This could be a hunger for
food.  It could be a hunger for knowledge.  More specific child terms
differentiate.]]></description>
            <pubDate>Fri, 05 Feb 2010 13:27:00 EST</pubDate>
            <guid
isPermaLink="true">http://idata.datacookbook.com/institution/terms/123
45</guid>
        </item>
    </channel>
</rss>
```

3.2 HTML Response Format

In an HTML response, the list of matching terms could be returned in a simple structure in HTML, with classes and names assigned so you could target CSS to fit their appearance to your application.

Sample HTML service transaction response:

```
<div class="IDATA_services">
    <div class="IDATA_DC_TermList">
        <div class="IDATA_DC_Term" name="hungry" id="12345">
            <div class="IDATA_DC_TermName">hungry</div>
            <div class="IDATA_DC_FunctionalDefinition">A hungry
student is one who has a need that he or she feels must be met.  This
could be a hunger for food.  It could be a hunger for knowledge.  More
specific child terms differentiate.</div>
            <div class="IDATA_DC_URL"><a
href="http://idata.datacookbook.com/terms/12345">http://idata.datacook
book.com/terms/12345</a></div>
        </div>
    </div>
</div>
```

3.3 XML Response Format

An XML response contains the same term information as the other options, but also includes more detail, including response status and message and potentially a re-cap of the request information. The XML below is a draft of an XML response for a lookup for "hungry". If no match is found, there will be either no <TermList> element or an empty <TermList> element in the response.

Sample XML service transaction response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceTransaction xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
    <ServiceResponse serviceName="term_lookup">
        <ResponseStatus>
            <ResponseCode>0</ResponseCode>
            <ResponseMessage>Success!</ResponseMessage>
        </ResponseStatus>
        <TermList>
            <Term>
                <name>hungry</name>
                <functional-definition>A hungry student is one who has
a need that he or she feels must be met.  This could be a hunger for
food.  It could be a hunger for knowledge.  More specific child terms
differentiate.</functional-definition>
                <perma-link-
url>http://idata.datacookbook.com/terms/12345</perma-link-url>
            </Term>
       </TermList>
    </ServiceResponse>
</ServiceTransaction>
```

3.4 JSON Response Format

JSON is generally used to support a Javascript AJAX implementation of API requests from within a browser. It uses the JavaScript call-back method of implementing cross-domain JSON AJAX requests because it is the most straightforward method of implementing cross-domain AJAX. In this method, we specify the name of a function that is invoked in our return JavaScript to tell the calling page to process the results of the request. The calling page is responsible for implementing that method. To invoke the API, the calling page implements the call-back method, then includes a javascript <script> tag with the API call as the URL. The API performs the search and returns JavaScript formatted like the sample below that invokes the call-back function. In this sample, the function the consumer would have to implement is named "processServiceResponse()". You specify the name of the callback function using the optional jsonFunction parameter.

The JSON response contains the same granularity and level of detail found in the XML response.

Sample JSON service transaction response:

```
processServiceResponse(
    {
        ServiceName : "term_lookup",
        ResponseStatus : {
            ResponseCode : 0,
            ResponseMessage : "Success!"
        },
        TermList : [
            {
                "version" : {
                    "name" : "hungry",
                    "term_functional_definition" : "A hungry student
is one who has a need that he or she feels must be met.  This could be
a hunger for food.  It could be a hunger for knowledge.  More specific
child terms differentiate.",
                    "perma_link_url" :
"http://idata.datacookbook.com/institution/terms/12345"
                }
            }
        ]
    }
);
```