WHERE 2 GET IT
LOCAL MARKETING FOR NATIONAL BRANDS

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

Where 2 Get It | **REST API Documentation v2.18**

Documentation Version Notes:

2.16 – Added Section on Ambiguous “multiple_address” search response
2.17 – Corrected a typo about how to use the “not equals” feature in the locatorsearch request
2.18 – Corrected several typos

WHERE 2 GET IT
LOCAL MARKETING FOR NATIONAL BRANDS

REST API DOCUMENTATION

V 2.18

Where 2 Get It, Inc.

714.660.4870

hello@where2getit.com

where2getit.com

WHERE 2 GET IT
LOCAL MARKETING FOR NATIONAL BRANDS

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

Where 2 Get It | **REST API Documentation v2.18**

I. **Introduction**

The Where 2 Get It REST API makes writing geospatial application extremely easy. All of the advanced geospatial functions are easily called by making a normal HTTP GET request. The response is an XML document. The basic flow of a Where 2 Get It REST API call is:

1) create the required XML structure,
2) URI encode it,
3) make a HTTP GET request,
4) then parse the return XML document.

Input XML Structure

All API calls are made to http://api.slippymap.com. The exact format of the HTTP GET request is http://api.slippymap.com/rest?xml_request=<uriencodedxml>. The pre-URI encoded xml will have a general format looking like:

```
<request>
        <appkey>YOURAPPKEY</appkey>
        <formdata id='keyword'>
                the specific input xml goes here
        </formdata>
</request>
```

A fully encoded URL would look like the following.
http://api.slippymap.com/rest?&xml_request=%3Crequest%3E%3Cappkey%3E79FCB260-BF9B-3C91-84AC-C8B0A9E5ED89%3C%2Fappkey%3E%3Cformdata%20id%3D%22locatorsearch%22%3E%3Cdataview%3Estore_default%3C%2Fdataview%3E%3Cgeolocs%3E%3Cgeoloc%3E%3Caddressline%3E60090%3C%2Faddressline%3E%3Clongitude%3E%3C%2Flongitude%3E%3Clatitude%3E%3C%2Flatitude%3E%3C%2Fgeoloc%3E%3C%2Fgeolocs%3E%3Cproximitymethod%3Estraightline%3C%2Fproximitymethod%3E%3Csearchradius%3E5%3C%2Fsearchradius%3E%3C%2Fformdata%3E%3C%2Frequest%3E

There are many utilities available to URI encode text.

Since this is a GET request, the request must stay within a total of 4K characters (2048 for IE).

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

II. Implementation

Since the use of the REST API involves your server(s) communicating with our servers, the setup is very simple.

- Obtain your appkey. The appkey is embedded in every request that is made by your application and validates that your server is allowed to make the request. Your receipt of an appkey involves sending us the domain name of the server(s) making the REST API requests and IP address(es) of the servers making the REST API requests. All REST API requests are validated against both the appkey and the domain name/ip address to ensure the request is coming from a valid source. You can send this information to your client services representative or email techsupport@where2getit.com.

- Use your appkey as part of the REST API call.

- Program you application to send/receive REST API calls and update your web content accordingly.

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

III. Reference

Node <appkey>

This node is mandatory and is a unique application key for this application. The appkey is a value provided to you either through your self-service sign up or from the Where 2 Get It client services department. The appkey is uniquely tied to the domain making the REST API call as well as the server(s) IP addresses.

Node <formdata>

This node is required for the definition of input logic. Valid id values are :

- geocoder – to geocode an input address
- reversegeocoder – to reverse geocode an input latitude/longitude
- drivingdirections – to perform a driving directions request
- geoip – to obtain a latitude/longitude and address information based upon IP address
- locatorsearch – to perform a proximity search
- productsearch – to perform a proximity search using a product as criteria
- etailersearch – to perform a search of online retailers based upon a product (separate document)
- getlist – to perform a search for retrieving a list of values

Formdata Structures

geocoder

The geocoder method supports both single and batch geocode requests. Each geocode request is embedded in a <geoloc> node. If the <country> element is not present, it will default to US.

The format of the XML data structure would look like any of the following.

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
      <addressline></addressline>
      <country></country> # ISO 2 digit country code or country name
    </geoloc>
  </geolocs>
</formdata>
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
      <address1></address1>
      <city></city>
      <state></state>  # state or province
      <country></country> # ISO 2 digit country code or country name
      <postalcode></postalcode>
    </geoloc>
  </geolocs>
</formdata>
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
      <address1></address1>
      <city></city>
      <province><province>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
      <country></country> # ISO 2 digit country code or country name
      <postalcode></postalcode>
    </geoloc>
  </geolocs>
</formdata>
```

The <country> element is the only "mandatory" input for a <geoloc> node. The <addressline> element can be used instead of the individual discreet elements <street>, <city>, <state>, <postalcode>. This makes it easier to do single line address input. If your data is already split up, you can use the discreet elements instead.
The use of the <street> element is still supported by we recommend using the <address1> element instead.
Likewise, the <region> element is still supported for international locations but we recommend using <province> instead.

The <type> and <badaddress> response elements can be ignored in this version of the service.
The <county> response element is only returned on city or postal code geocode requests.
The <georesult> response element is an indicator of the quality of the geocode result. The higher the first two digits, the better the result.

10 = Address level latitude/longitude (is also used for an intersection result)
09 = Street level latitude/longitude. The exact house number was not able to be found but the latitude/longitude is on the street specified in the request.
05 = Postalcode centroid latitude/longitude.
04 = City centroid latitude/longitude
02 = State centroid latitude/longitude
01 = Country centroid latitude/longitude

Examples (the <appkey> and outer <request> node are not shown for brevity)

Single address US geocode call (city centroid)

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
      <city>Chicago</city>
      <state>IL</state> # state or province
      <country>US</country>
    </geoloc>
  </geolocs>
</formdata>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<response code="1">
<collection name="address" count="1">
<address>
 <address1></address1>
 <address2></address2>
 <badaddress></badaddress>
 <city>Chicago</city>
 <country>US</country>
 <county>Cook</county>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
 <georesult>04 WORA_EXACT</georesult>
 <latitude>41.88414</latitude>
 <longitude>-87.63238</longitude>
 <objectuid></objectuid>
 <postalcode>60602</postalcode>
 <province></province>
 <state>IL</state>
 <type></type>
</address>
</collection>
</response>
```

Single address US geocode call (city centroid)) – alternative method using single line address

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
      <addressline>Chicago, IL</addressline>
    </geoloc>
  </geolocs>
</formdata>
```

Single address US geocode call (street level geocode)

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
          <address1>111 Main Street</address1>
          <city>Chicago</city>
          <state>IL</state>
          <country>US</country>
    </geoloc>
  </geolocs>
</formdata>
```

Single address US geocode call (street level geocode) – alternative method using single line address

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
          <addressline>111 Main Street, Chicago, IL</addressline>
    </geoloc>
  </geolocs>
</formdata>
```

Single address US geocode call (zip code centroid)

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
          <postalcode>60090</postalcode>
          <country>US</country>
    </geoloc>
  </geolocs>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
</formdata>
Response
<?xml version="1.0" encoding="UTF-8"?>
<response code="1">
<collection name="address" count="1">
<address>
 <address1></address1>
 <address2></address2>
 <badaddress></badaddress>
 <city>Wheeling</city>
 <country>US</country>
 <county>Cook</county>
 <georesult>05 WORA_EXACT</georesult>
 <latitude>42.1338</latitude>
 <longitude>-87.9295</longitude>
 <objectuid></objectuid>
 <postalcode>60090</postalcode>
 <province></province>
 <state>IL</state>
 <type></type>
</address>
</collection>
</response>
```

Single address US geocode call (zip code centroid) – alternative method using single line address

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
        <addressline>60090</addressline>
    </geoloc>
  </geolocs>
</formdata>
```

Single address Canadian geocode call (city centroid)

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
        <city>Toronto</city>
        <country>CA</country>
    </geoloc>
  </geolocs>
</formdata>
```

Single address German geocode call (city centroid)

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
        <city>Frankfurt</city>
        <country>DE</country>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
    </geoloc>
  </geolocs>
</formdata>
```

Single address France geocode call (street level geocode)

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
        <address1>Rue Cler</address1>
        <city>Paris</city>
        <country>FR</country>
    </geoloc>
  </geolocs>
</formdata>
```

Multiple address US geocode call (street level geocode)

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
        <address1>363 Milano Dr</address1>
        <city>Cary</city>
        <state>IL</state>
        <country>US</country>
    </geoloc>
    <geoloc>
        <address1>9825 Capitol Drive</address1>
        <city>Wheeling</city>
        <state>IL</state>
        <country>US</country>
    </geoloc>
  </geolocs>
</formdata>
<?xml version="1.0" encoding="UTF-8"?>
<response code="1">
<collection name="address" count="2">
<address>
  <address1>363 MILANO DR</address1>
  <address2></address2>
  <badaddress></badaddress>
  <city>CARY</city>
  <country>US</country>
  <county></county>
  <georesult>10 ORAADDRESS</georesult>
  <latitude>42.20347493456848</latitude>
  <longitude>-88.26325199350477</longitude>
  <objectuid></objectuid>
  <postalcode>60013</postalcode>
  <province></province>
  <state>IL</state>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
 <type></type>
</address>
<address>
 <address1>9825 CAPITOL DR</address1>
 <address2></address2>
 <badaddress></badaddress>
 <city>WHEELING</city>
 <country>US</country>
 <county></county>
 <georesult>10 ORAADDRESS</georesult>
 <latitude>42.11107309259042</latitude>
 <longitude>-87.90894127160638</longitude>
 <objectuid></objectuid>
 <percent>0.7204472843450479</percent>
 <postalcode>60090</postalcode>
 <province></province>
 <state>IL</state>
 <type></type>
</address>
</collection></response>
```

Multiple address US geocode call (street level geocode) – alternative method using single line address

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
        <addressline>363 Milano Dr, Cary, IL</addressline>
    </geoloc>
    <geoloc>
        <addressline>9825 Capitol Drive, Wheeling, IL</addressline>
    </geoloc>
  </geolocs>
</formdata>
```

Multiple address US geocode call (street level geocode with multiple address response)

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
        <address1>543 Hough Street</address1>
        <city>Barrington</city>
        <state>IL</state>
        <country>US</country>
    </geoloc>
    <geoloc>
        <address1>9825 Capitol Drive</address1>
        <city>Wheeling</city>
        <state>IL</state>
        <country>US</country>
    </geoloc>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
  </geolocs>
</formdata>
```

Ambiguous address US geocode call (street level geocode) – alternative method using single line address

```
<formdata id='geocoder'>
  <geolocs>
    <geoloc>
        <addressline>543 Hough Street, Barrington, IL</addressline>
    </geoloc>
  </geolocs>
</formdata>
Response
<?xml version="1.0" encoding="UTF-8"?>
<response code="1">
<collection name="address" count="1">
<collection name="multiple_address" count="2"><address>
 <address1>543 S HOUGH ST</address1>
 <address2></address2>
 <badaddress></badaddress>
 <city>BARRINGTON</city>
 <country>US</country>
 <county></county>
 <georesult>10 ORAADDRESS</georesult>
 <latitude>42.14950256410256</latitude>
 <longitude>-88.13606564102564</longitude>
 <objectuid></objectuid>
 <postalcode>60010</postalcode>
 <province></province>
 <state>IL</state>
 <type></type>
</address>
<address>
 <address1>543 N HOUGH ST</address1>
 <address2></address2>
 <badaddress></badaddress>
 <city>BARRINGTON</city>
 <country>US</country>
 <county></county>
 <georesult>10 ORAADDRESS</georesult>
 <latitude>42.16051</latitude>
 <longitude>-88.1360675</longitude>
 <objectuid></objectuid>
 <postalcode>60010</postalcode>
 <province></province>
 <state>IL</state>
 <type></type>
</address>
</collection>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
</collection>
</response>
```

reversegeocoder

The reverse geocode method supports both single and batch geocode requests. Each reverse geocode request is embedded in a <geoloc> node. The <country> node is necessary but will slightly improve performance if provided.

The format of the XML data structure would look like the following:

```
 <formdata id='reversegeocoder'>
  <geolocs>
    <geoloc>
      <country></country> # ISO 2 digit country code or country name
      <latitude></latitude>
      <longitude</longitude>
    </geoloc>
  </geolocs>
</formdata>
```

Examples (the <appkey> and outer <request> node are not shown for brevity)

Multiple address US reverse geocode call

```
<formdata id='reversegeocoder'>
  <geolocs>
    <geoloc>
          <latitude>33.8502</latitude>
          <longitude>-117.7912</longitude>
          <country>US</country>
    </geoloc>
    <geoloc>
          <latitude>42.1919</latitude>
          <longitude>-87.9257</longitude>
          <country>US</country>
    </geoloc>
  </geolocs>
</formdata>
```

Single address Belgium reverse geocode call

```
<formdata id='reversegeocoder'>
  <geolocs>
    <geoloc>
          <latitude>51.220581</latitude>
          <longitude>4.399722</longitude>
          <country>BE</country>
    </geoloc>
  </geolocs>
</formdata>
```

locatorsearch

The proximity search method, “locatorsearch” can perform either straight-line or drive time/distance searches. A <geoloc> node is required as the starting point for the proximity search. The <geoloc> node should contain

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

either an address to be geocoded or a latitude/longitude coordinate but not both. If both are present, the latitude/longitude will take precedence for the center point of the search. The response from a “locatorsearch” method will be a collection of locations from your database. A straightline (as the crow flies) proximity search is the default method unless it is overridden in the request or the account setup has specified a different proximity search method. The use of the the “locatorsearch” function is only applicable if Where2GetIt hosts your location database. We are not able to do a proximity search on databases inside your infrastructure.

```
<formdata id='locatorsearch'>
  <geolocs>
    <geoloc> # supply either an address or lat/lon, but not both
      <addressline></addressline> # use this short hand for a single line address input
      <latitude></latitude>
      <longitude</longitude>
      <country></country>
    </geoloc>
  </geolocs>
  <where></where>
  <dataview></dataview> #defaults to all fields in your database
  <limit></limit>
  <order></order>
  <subsets>
    <subset>
      <limit></limit>
      <column></column>
      <value></value>
    </subset>
  </subsets>
  <events>
    <where></where>
  </events>
  <proximitymethod>straightline|drivetime</proximitymethod>
  <sort>distance|time</sort>
  <cutoff>(numerical value)</cutoff>
  <cutoffuom>mile|meter|KM|second|minute|hour</cutoffuom>
  <timeuom>second|minute|hour</timeuom>
  <searchradius>(numerical value)</searchradius>
  <radiusuom>mile|meter|KM</radiusuom>
  <route>
    <routepreference>shortest|fastest</routepreference>
    <roadpreference>highway|local</roadpreference>
  </route>
  <map>
    <size></size>
    <icons>
      <default>
        <offset></offset>
        <path></path>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
        <label>
          <color></color>
          <size></size>
          <font></font>
          <offset></offset>
        </label>
      </default>
      <center_marker></center_marker>
    </icons>
  </map>
</formdata>
```

Elements

<limit>
This element will limit the number of locations returned in the response xml. This is a numeric value such as 4.
<order>
This sorts the response using the database columns specified. By default, the response is sorted by distance. Multiple columns can be specified by separating with a comma. Example, <order>rank,_distance</order>
<proximitymethod>
This can be either “straightline” or “drivetime”. If “straightline” is used, the results will automatically be sorted in distance order. Likewise, the <route> node will be ignored.
<cutoff>
This element can be used to limit the results. It is a numerical value and the unit of measure is defined by the <cutoffuom> element. It is particularly used when specifying a “drivetime” call. You may specify a search radius of 25 miles but cutoff the results using 10 minutes. This value can either be stored in your profile or provided in the call.
<cutoffuom>
This element defines the unit of measure for the <cutoff> value. This value can either be stored in your profile or provided in the call. If the <cutoffuom> is mile|meter|km, then a drive distance proximity search is performed. If the <cutoffuom> is second|minute|hour, then a drive time proximity search is performed.
<searchradius>
This element defines the initial proximity search radius. This limits the locations selected from the hosted database. An initial proximity search using a straight-line method is done using this value. If the “drivetime” method is specified in the call, an additional drive time analysis is done on the locations derived by the initial search to arrive at the final results. This value can either be stored in your profile or provided in the call.
<radiusuom>
This element defines the unit of measure for the search radius. This value can either be stored in your profile or provided in the call.
<routepreference>
This element is used if the “drivetime” method is used. This value can either be stored in your profile or provided in the call.
<roadpreference>
This element is used if the “drivetime” method is used. This value can either be stored in your profile or provided in the call.
<dataview>
This element is used to provide an alternate set of database fields. All database fields are returned by default. This is an optional element.

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

<subsets><subset>
This element is used to limit the number of results based upon a specific column value. As an example, you could say only show 2 locations that have a value of 99 in the rank column.
<subsets><subset><limit>
This element specifies the numerical limit for the subset of data.
<subsets><subset><column>
This element specifies the column to use for the criteria of this subset.

<subsets><subset><value>
This element specifies the value to use in matching the subset criteria.
<events>
This element is used to get events associated with each location based upon the criteria specified in the <where> clause. The events are returned as a separate collection after the locations with each having a pointer to the location they reference. Use the <where> clause to include only those events required. Example <where><eventdate><ge>now()</ge></eventdate></where>
<map>
This element is used to generate a static map url in the response xml. The static map url can then be used as the source for an image tag to embed the map on a web page.
<map><size>
This element will dictate what size of image will be generated for the static map. Use 2 numerical values separated by a comma. Example, <size>750,500</size>
<map><icons>
This element begins the definition of what icons should appear on the static map. Each icon name will have a separate structure.
<map><icons><default>
This element would be the minimum requirement to specify a default icon. Without this, a colored dot will appear on the map without numbering. The value “default” coincides with the “icon” column value for each location. During the file upload, if you submitted an icon column to have custom icons on a per location basis, you would add a new element for each custom icon. Example, <map><icons><rockport> would begin the definition of a custom icon if I had locations using this value I the “icon” column.
<map><icons><default><path>
This element defines the path on the W2GI server where to get the icon image. By default, it will use icons named in the global W2GI images directory. If it is a custom icon found in just your client directory, the client services staff can help you with the exact path. Example, <path>icons/pin1-shadow.png</path>
<map><icons><default><offset>
This element defines the positioning of the icon in relationship to the latitude/longitude of the location. The numbers are in pixels. Example, <offset>36,13</offset>
<map><icons><default><label>
This element starts the definition of the icon label. You can specify the font size, the font type, font color and the positioning of the label on the icon.
<map><icons><default><label><color>
This element defines the font color to use for the icon label. Use a 6 digit hexadecimal value. Example, <color>FFFFFF</color>
<map><icons><default><label><size>
This element defines the font size for the icon label. Example, <size>10</size>
<map><icons><default><label>

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

This element defines the type of font to use for the icon label. Example, arial-bold
<map><icons><default><label><offset>
This element defines the positioning of the label inside the icon. The numbers are in pixels. Example, <offset>15,11</offset>
<map><icons><center_marker>
This element defines the type of center marker to use on the map. There is no center marker by default. Example, <center_marker><path>red-star.png</path></center_marker>

<where>
This node defines the criteria logic (if applicable) to any database fields. The database fields must match the element names to create the criteria. The element names are essentially the database column names in your database. There are 13 basic operator elements that can be used to create a criteria search. The complexity of the <where> node can be quite great as unlimited criteria is supported.

- *<and>* – all subsequent nodes must evaluate to true for the criteria to be true
- *<or>* – one of the subsequent nodes must evaluate to true for the criteria to be true
- *<like>* – string matching – the input field will be string matched to determine if the database field contains the input string
- <ilike> - string matching – the input field will be string matched (case insensitive) to determine if the database field contains the input string.
- *<notlike>* – string matching – the input field will be string matched to determine if the database field does not contain the input string
- *<in>* – the comma delimited set of values from a selection list or multiple checkboxes will be used to determine if the database fields matches any of these values
- *<notin>* – the comma delimited set of values from a selection list or multiple checkboxes will be used to determine if the database fields do not match any of the values
- *<eq>* – Equals – the input field must exactly match the database value
- *<ne>* – Not equals – the input field does not exactly match the database value
- *<gt>* – Greater than
- *<lt>* – Less than
- *<ge>* – Greater than or equal to
- *<le>* – Less than or equal to
- *<between>* – The database field value will fall between the two input values. Use comma separator.

The XML response will be a collection of locations with each location node containing the database elements defined by the dataview.

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

Examples (the <appkey> and outer <request> node are not shown for brevity)

Basic straight-line proximity search with criteria

```
<formdata id='locatorsearch'>
        <geolocs>
                <geoloc>
                        <addressline>60090</addressline>
                        <country>US</country>
                </geoloc>
        </geolocs>
        <where>
                <and>
                        <or>
                                <productname>
                                        <like>a%</like>
                                </productname>
                                <category>
                                        <in>video,audio</in>
                                </category>
                        </or>
                        <store>
                                <eq>Circuit City</eq>
                        </store>
                </and>
        </where>
</formdata>
```

The sql where clause would get constructed like the following.

where (((productname like 'a%') or (category in ('video','audio')) and store = 'Circuit City')

Basic straight-line proximity search with no criteria – 25 mile search radius

```
<formdata id='locatorsearch'>
        <geolocs>
                <geoloc>
                        <addressline>60090</addressline>
                        <country>US</country>
                </geoloc>
        </geolocs>
        <searchradius>25</searchradius>
</formdata>
```

Drive time proximity search with no criteria – 25 KM search radius, 10 minute cutoff using shortest driving distance with highway preference

```
<formdata id='locatorsearch'>
        <geolocs>
                <geoloc>
                        <addressline>Roma</addressline>
                        <country>IT</country>
                </geoloc>
        </geolocs>
        <searchradius>25</searchradius>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
        <radiusuom>KM</radiusuom>
        <proximitymethod>drivetime</proximitymethod>
        <sort>time</sort>
        <cutoff>10</cutoff>
        <cutoffuom>minute </cutoffuom>
        <route>
                <routepreference>shortest</routepreference>
                <roadpreference>highway</roadpreference>
        </route>
</formdata>
```

Drive distance proximity search with no criteria – 10 mile search radius, 5 mile cutoff using fastest driving distance with highway preference

```
<formdata id='locatorsearch'>
        <geolocs>
                <geoloc>
                        <addressline>9825 Capitol Drive, Wheeling, IL</addressline>
                        <country>US</country>
                </geoloc>
        </geolocs>
        <searchradius>10</searchradius>
        <radiusuom>mile</radiusuom>
        <proximitymethod>drivetime</proximitymethod>
        <sort>distance</sort>
        <cutoff>5</cutoff>
        <cutoffuom>mile</cutoffuom>
        <route>
                <routepreference>fastest</routepreference>
                <roadpreference>highway</roadpreference>
        </route>
</formdata>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

Ambiguous "multiple_address" search response

In some instances when conducting a locatorsearch or other proximity search request, a "multiple_address" result may be returned. This means that there were multiple matches for the given search input. Typically this occurs when there are two or more cities with the same name within the same state (or if it was an international search, cities that share the same name within the given country). To handle this response properly in your application, you should display these results to the user and prompt them to confirm which area the meant and resubmit the search either with the postalcode appended or lat/long coordinates.

Example Response:

```
<response code="1">
<collection name="multiple_address" count="2" country="US" city="Rome" radius="50" radiusuom="mile" posta
lcode="13440" province="" address="" centerpoint="-75.45336,43.21516"state="NY">
<address>
<address1/>
<address2/>
<badaddress/>
<city>Rome</city>
<country>US</country>
<county>Essex</county>
<georesult>04 WORA_EXACT</georesult>
<latitude>44.4406028</latitude>
<llpoint/>
<longitude>-73.6881949</longitude>
<objectuid/>
<postalcode>12912</postalcode>
<province/>
<state>NY</state>
<type/>
</address>
<address>
<address1/>
<address2/>
<badaddress/>
<city>Rome</city>
<country>US</country>
<county>Oneida</county>
<georesult>04 WORA_EXACT</georesult>
<latitude>43.21516</latitude>
<llpoint/>
<longitude>-75.45336</longitude>
<objectuid/>
<postalcode>13440</postalcode>
<province/>
<state>NY</state>
<type/>
</address>
</collection>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

</response>

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

drivingdirections

The "drivingdirections" method is used to get turn-by-turn directions. A minimum of two <geoloc> nodes is required. Each <geoloc> node should contain either an address to be geocoded or a latitude/longitude coordinate but not both. If both are present, the latitude/longitude will take precedence. If more than two <geoloc> nodes are provided, turn-by-turn directions will be provided to each subsequent location (in the order they are provided). The response from a "drivingdirections" method will be a collection of maneuvers and/or segment geometries.

It is highly preferred to use the latitude/longitude values for the driving direction end points. This can be accomplished by geocoding the addresses first. If you do pass in an address as part of the "drivingdirection" request, be prepared to handle ambiguous address responses.

```
<formdata id='drivingdirections'>
  <geolocs> # the first geoloc will be the starting address
    <geoloc> # starting location -- supply either an address or lat/lon
      <addressline></addressline> # use this short hand for a single line address input
      <address1></address1>
      <city></city>
      <state></state> # state
      <province></province> # province
      <country></country> # ISO 2 digit country code
      <postalcode></postalcode>
      <latitude></latitude>
      <longitude</longitude>
    </geoloc>
    <geoloc> # destination location -- supply either an address or lat/lon
      <addressline></addressline> # use this short hand for a single line address input
      <address1></address1>
      <city></city>
      <state></state> # state
      <province></province> # province
      <country></country> # ISO 2 digit country code or country name
      <postalcode></postalcode>
      <latitude></latitude>
      <longitude</longitude>
    </geoloc>
    <geoloc> # next destination location -- supply either an address or lat/lon
      <addressline></addressline> # use this short hand for a single line address input
      <address1></address1>
      <city></city>
      <state></state> # state
      <province></province> # province
      <country></country> # ISO 2 digit country code or country name
      <postalcode></postalcode>
      <latitude></latitude>
      <longitude</longitude>
    </geoloc>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
  </geolocs>
  <route>
    <language>english|spanish|french|german|italian</language
    <routepreference>shortest|fastest</routepreference>
    <roadpreference>highway|local</roadpreference>
    <returndrivingdirections>true|false</returndrivingdirections>
    <returnroutegeometry>true|false</returnroutegeometry>
    <returnsegmentgeometry>true|false</returnsegmentgeometry>
    <timeuom>second|minute|hour</timeuom>
    <distanceuom>mile|meter|KM</distanceuom>
    <atw>
       <radius></radius>
       <radiusuom></radiusuom>
       <poitype></poitype> #defaults to locator stores/dealers
    </atw>
  </route>
  <map>
    <size></size>
    <icons>
      <start>
        <path></path>
        <size></size>
      </start>
      <end>
        <path></path>
        <size></size>
      </end>
    </icons>
  </map>
</formdata>
```

Elements

<language>
This element defines the language used to display the turn-by-turn maneuvers. The default is “English”. This value can either be stored in your profile or provided in the call. Five major languages are supported today, English, French, German, Spanish and Italian.
<routepreference>
This element determines whether the fastest or shortest path is calculated. This value can either be stored in your profile or provided in the call. The default is “fastest”.
<roadpreference>
This element determines the type of road to use in the path calculation. This value can either be stored in your profile or provided in the call. The default is “highway”.
<returndrivingdirections>
This element controls whether the textual turn-by-turn maneuvers are returned. The default is “true”. This value can either be stored in your profile or provided in the call.
<returnroutegeometry>

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

This element controls whether the entire route segments (defined by latitude/longitude pairs) are returned. The default is “false”. This value can either be stored in your profile or provided in the call.
<returnsegmentgeometry>
This element controls whether the individual maneuver segments (defined by latitude/longitude pairs) are returned. The default is “false”. This value can either be stored in your profile or provided in the call.
<distanceuom>
If this element is present, it will format the output distance to this unit of measure. Defaults to mile.
<timeuom>
If this element is present, it will format the output time to this unit of measure. Defaults to minute.
<atw>
If this element is present, it will produce store/dealer locations along the driving direction route.
<atw><radius>
This element controls how far from the driving direction route to look for store dealer locations.
<atw><radiusuom>
This element specifies the unit of measurement for the along the way radius.
<map>
This element is used to generate a static map url in the response xml. The static map url can then be used as the source for an image tag to embed the map on a web page.
<map><size>
This element will dictate what size of image will be generated for the static map. Use 2 numerical values separated by a comma. Example, <size>750,500</size>
<map><icons>
This element begins the definition of the start/end icons that will appear on the static map. Each icon will have a separate structure.
<map><icons><start>
This element begins the definition of the icon used for the start of the driving directions.
<map><icons><end>
This element begins the definition of the icon used for the end of the driving directions.
<map><icons><start><path>
This element defines the path on the W2GI server where to get the icon image. By default, it will use icons named in the global W2GI images directory. If it is a custom icon found in just your client directory, the client services staff can help you with the exact path. One is required for both the start and end icons. Example, <path>icons/pin1-shadow.png</path>
<map><icons><start><size>
This element defines the size of the icon in pixels. Example, <size>30</size>

Examples (the <appkey> and outer <request> node are not shown for brevity)
Point to point driving directions using default route parameters

```
<formdata id='drivingdirections'>
        <geolocs> # the first geoloc will be the starting address
                <geoloc>
                        <addressline>543 N. Hough Street, Barrington, IL</addressline>
                        <country>US</country>
                </geoloc>
                <geoloc>
                        <country>US</country>
                        <addressline>60090</addressline>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
            </geoloc>
        <geolocs>
</formdata>
```

Point to point driving directions in Spanish using shortest route with local road preferences

```
<formdata id='drivingdirections'>
        <geolocs> # the first geoloc will be the starting address
            <geoloc>
                <addressline>Madrid</addressline>
                <country>ES</country>
            </geoloc>
            <geoloc>
                <addressline>Venice</addressline>
                <country>IT</country>
            </geoloc>
        <geolocs>
        <route>
            <language>spanish</language
            <routepreference>shortest</routepreference>
            <roadpreference>local</roadpreference>
        </route>
</formdata>
```

Multipoint driving directions

```
<formdata id='drivingdirections'>
        <geolocs> # the first geoloc will be the starting address
            <geoloc>
                <addressline>9825 Capitol Drive, Wheeling, IL</addressline>
                <country>US</country>
            </geoloc>
            <geoloc>
                <addressline>543 N. Hough Street, Barrington, IL</addressline>
                <country>US</country>
            </geoloc>
            <geoloc>
                <addressline>363 Milano Drive, Cary, IL</addressline>
                <country>US</country>
            </geoloc>
        <geolocs>
</formdata>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

geoip

The “geoip” method is used to get the latitude/longitude for a specified IP address. A single or multiple IP addresses can be submitted in a request. The data is best covered in the US down to a city/postal code area. Countries outside of the US have less detail but would be able to identify at least the country.

Sample request

```
<formdata id='geoip'>
        <geolocs>
                <geoloc>
                        <ipaddress>216.64.204.203</ipaddress>
                </geoloc>
                <geoloc>
                        <ipaddress>64.236.91.21</ipaddress>
                </geoloc>
                <geoloc>
                        <ipaddress>192.193.217.120</ipaddress>
                </geoloc>
        </geolocs>
</formdata>
```

The response would look like

```
<?xml version="1.0" encoding="UTF-8"?>
<response code="1">
<collection name="geoip" count="3"><geoip>
 <city>Wheeling</city>
 <country>US</country>
 <ipaddress>216.64.204.203</ipaddress>
 <latitude>42.1286</latitude>
 <longitude>-87.9237</longitude>
 <postalcode>60090</postalcode>
 <region>IL</region>
</geoip>
<geoip>
 <city>Reston</city>
 <country>US</country>
 <ipaddress>64.236.91.21</ipaddress>
 <latitude>38.9311</latitude>
 <longitude>-77.3489</longitude>
 <postalcode>20191</postalcode>
 <region>VA</region>
</geoip>
<geoip>
 <city>New York</city>
 <country>US</country>
 <ipaddress>192.193.217.120</ipaddress>
 <latitude>40.7214</latitude>
 <longitude>-74.0052</longitude>
 <postalcode>10013</postalcode>
 <region>NY</region>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
</geoip>
</collection>
</response>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

productsearch

The “productsearch” method is very similar to the “locatorsearch” method. The difference is that with the “productsearch” method, a product sku must be provided. A <geoloc> node is required as the starting point for the proximity search. The <geoloc> node should contain either an address to be geocoded or a latitude/longitude coordinate but not both. If both are present, the latitude/longitude will take precedence for the center point of the search. The response from a “productsearch” method will be a collection of locations from your database that carry the specific product referenced by the sku parameter. The use of the the “locatorsearch” function is only applicable if Where2GetIt hosts your location database. We are not able to do a proximity search on databases inside your infrastructure.

Sample request

```
<request>
   <appkey>455C968C-C391-3868-90CF-1993F54CFD32</appkey>
   <geoip>1</geoip>
   <formdata id='productsearch'>
      <geolocs><geoloc>
        <addressline>92807</addressline>
        <latitude></latitude>
        <longitude></longitude>
        <country></country>
      </geoloc></geolocs>
      <order>rank,_distance</order>
      <searchradius>5|10|20|40|60</searchradius>
      <where>
          <sku><eq>TM4111</eq></sku>
      </where>
   </formdata>
</request>
```

Sample response

```
<?xml version="1.0" encoding="UTF-8"?><response code="1">
<collection name="poi" count="1" country="US"
 city="Anaheim" postalcode="92807" province="" address="" centerpoint="-117.7912,33.8502"
state="CA"><poi>
  <name>R E I Santa Ana</name>
  <_distance>8.79</_distance>
  <accessories>1</accessories>
  <activeflag>1</activeflag>
  <address1>1411 Village Way</address1>
  <address2></address2>
  <asin></asin>
  <category1>Fall 08</category1>
  <category2></category2>
  <category3></category3>
  <city>Santa Ana</city>
  <clientkey>16104</clientkey>
  <country>US</country>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

<currency>USD</currency>
<dealerid>4098</dealerid>
<description>More fun in the sun thanks to UPF protection in a functional, durable fabric. These shorts are quick drying and have pockets to stash cash, cards and keys.</description>
<email></email>
<equipment>0</equipment>
<fax></fax>
<fishing>0</fishing>
<footwear>1</footwear>
<hunting>0</hunting>
<icon>default</icon>
<image></image>
<latitude>33.7311800562351</latitude>
<longitude>-117.835884756549</longitude>
<map></map>
<msrp>34.95</msrp>
<outerwear>1</outerwear>
<phone>(714) 543-4142</phone>
<pname>Omni-Dry® Silver Ridge™ Cargo Short</pname>
<postalcode>92705-4714</postalcode>
<province></province>
<puid>85320</puid>
<rank>50</rank>
<sku>TM4111</sku>
<sportswear>1</sportswear>
<state>CA</state>
<storenumber>17</storenumber>
<uid>935598291</uid>
<upc></upc>
<url></url>
<youth>0</youth>
</poi>
</collection></response>

getlist

The “getlist” method is used to lists of values from a database table. This is normally used to get a list of US states or a list of countries. The “softmatch” attribute can be used inconjunction with the <ilike> operator in the where clause to do softmatching against database columns. You could also just put the % symbols in the text input. Use the <eq> operator for equality matching.

Sample request to get a list of cities for a specific zip code

```
<request>
        <appkey>455C968C-C391-3868-90CF-1993F54CFD32</appkey>
        <formdata id='getlist'>
                <objectname>City</objectname>
                <where>
                        <zip><eq>60173</eq></zip>
                </where>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
        </formdata>
</request>
```

The response would look like

```
<?xml version="1.0" encoding="UTF-8"?>
<response code="1">
        <collection name="city" count="3">
                <city>
                        <city>HOFFMAN EST</city>
                        <country>US</country>
                        <fips>17031</fips>
                        <latitude>42.0523</latitude>
                        <longitude>-88.0478</longitude>
                        <msa>1602</msa>
                        <state>IL</state>
                        <zip>60173</zip>
                        </city>
                        <city>
                        <city>HOFFMAN ESTATES</city>
                        <country>US</country>
                        <fips>17031</fips>
                        <latitude>42.0523</latitude>
                        <longitude>-88.0478</longitude>
                        <msa>1602</msa>
                        <state>IL</state>
                        <zip>60173</zip>
                        </city>
                        <city>
                        <city>SCHAUMBURG</city>
                        <country>US</country>
                        <fips>17031</fips>
                        <latitude>42.0523</latitude>
                        <longitude>-88.0478</longitude>
                        <msa>1602</msa>
                        <state>IL</state>
                        <zip>60173</zip>
                </city>
        </collection>
</response>
```

An example of a getlist request to get a specific store location.

```
<request>
        <appkey>E44D00DE-57DE-11DF-9086-D8145B67B72A</appkey>
        <formdata id='getlist'>
                <objectname>Locator::Store</objectname>
                <where>
                        <clientkey><eq>416</eq></clientkey>
                </where>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com

```
        </formdata>
</request>
```

An example of a getlist request to get store locations by name (using softmatch).

```
<request>
        <appkey>E44D00DE-57DE-11DF-9086-D8145B67B72A</appkey>
        <formdata id='getlist' softmatch='1'>
                <objectname>Locator::Store</objectname>
                <where>
                        <name><ilike>Phoenix</ilike></name>
                </where>
        </formdata>
</request>
```

An example of a getlist request to get store locations by name (using softmatch but without the attribute).

```
<request>
        <appkey>E44D00DE-57DE-11DF-9086-D8145B67B72A</appkey>
        <formdata id='getlist'>
                <objectname>Locator::Store</objectname>
                <where>
                        <name><ilike>Phoenix%</ilike></name>
                </where>
        </formdata>
</request>
```

Where 2 Get It, Inc.
714.660.4870
hello@where2getit.com
where2getit.com