Reference Guide

WebSmart
Street Search

MELISSA DATA®

WebSmart
Street Search

Reference Guide

Melissa Data Corporation

Copyright

Trademarks

Melissa Data Corporation

22382 Avenida Empresa
Rancho Santa Margarita, CA 92688-2112

Phone: 1-800-MELISSA (1-800-635-4772)
Fax: 949-589-5211

E-mail: info@MelissaData.com
Internet: www.MelissaData.com

For the most recent version of this document, visit
http://www.melissadata.com/

Document Code: DQTWSSSRG
Revision Number: 18102013.15

Dear Developer,

I would like to take this opportunity to thank you for your interest in Melissa Data products and introduce you to the company.

Melissa Data has been a leading provider of data quality and address management solutions since 1985. Our data quality software, Cloud services, and data integration components verify, standardize, consolidate, enhance and update U.S., Canadian, and global contact data, including addresses, phone numbers, and email addresses, for improved communications and ROI. More than 5,000 companies rely on Melissa Data to gain and maintain a single, accurate and trusted view of critical information assets.

This manual will guide you through the functions of our easy-to-use programming tools. Your feedback is important to me, so please don't hesitate to email your comments or suggestions to me at: Ray@MelissaData.com.

I look forward to hearing from you.

Best Wishes,

Raymond F. Melissa

President/CEO

Table of Contents

1 Welcome to WebSmart Services

The WebSmart Services are a collection of services that can be accessed by any application, allowing you to incorporate Melissa Data's technology into your programs without worrying about continually downloading and installing updates.

Melissa Data currently offers the following services:

- **Address Verifier** — Verify and standardize one or more mailing address. This service also appends ZIP + 4® and Carrier Route information.
- **Email Verifier** — Verify, correct and update, domain names from one or more email addresses.
- **GeoCoder** — Returns geographic, census, and demographic data for almost any location in the United States. Uses multisource data to return latitude and longitude down to rooftop accuracy of over 95% of all physical addresses in the United States.
- **IP Locator** — Returns name and geographic information for the owner of a public IP address.
- **Delivery Indicator** — Indicates whether an address represents a business or residential address.
- **Name Parser** — Parses and genderizes personal names and also generates salutations for correspondence.

- **Street Search** — Searches a ZIP Code™ from street address ranges matching a specific pattern and, optionally, a street number.
- **ZIP Search** — Matches city names with ZIP/Postal codes, ZIP/Postal codes with city names and searches for city names matching a pattern with a given state.
- **Phone Verifier** — Verifies and parses phone numbers, as well as identifying phone numbers as residential, business, VOIP or wireless.
- **Property** — Returns basic or detailed information about the size, ownership, and structures on a given parcel of land.

Both GeoCoder and Delivery Indicator work from an “address key” returned by the Address Verifier service, therefore, an address must first be submitted to the Address Verifier before you can use either of the other two services.

There are three ways to access the WebSmart Services:

- **SOAP** — The SOAP interface allows you to add the Web Service to an application as if it were a component object or DLL. You can then access the Web Service elements and execute commands as if they were properties and methods.
- **XML** — The Web Service can also submit a request as an XML document. It will then return the processed records as another XML document that can be parsed using whatever XML tools you utilize in your development environment.
- **REST** — This interface allows you to submit a single address record as part of a URL string and returns the processed record as an XML document identical to the one returned by the XML interface.

Using the REST service may require that you encode certain characters using the proper URL entities before adding them to a URL. Characters like spaces, slashes, ampersands and others must be replaced by special codes, which usually consist of a percent sign followed by a two-digit hexadecimal number.

The following table shows the replacements for the most common characters.

Character	URL Encoded
Space	%20 or +
*	%2A
#	%23
&	%26
%	%25

Character	URL Encoded
$	%28
+	%2B
,	%2C
/	%2F
:	%3A
;	%3B
<	%3C
=	%3D
>	%3E
?	%3F
@	%40
[	%5B
]	%5D
~	%7E

Many modern programming languages have a URL encode and URL decoding function that automates these character replacements.

Special Characters

Because the WebSmart Services are XML-based, certain characters cannot be passed as data. They would be interpreted as part of the XML structure and would cause errors. The following codes must be substituted for these characters.

Character	URL Encoded
&	& (ampersand)
“	" (left/right quotes should be replaced with straight quotes)
‘	' (apostrophe)
<	< (less-than)
>	> (greater-than)

2 An Introduction to Street Search

The Web Smart Street Search service matches a street name or just a partial street name and returns any valid addresses that match that pattern.

Used in conjunction with the Web Smart Address Verifier, Street Search can be used to match incorrect or misspelled street names, list possible ranges for a street or a highrise and generate probable suggestions for end users to select. For example, if “123 Main” matches both “123 Main St” and “123 Main Ave” in the same ZIP Code, the Address Verifier would return a multiple match error. Street Search can return all records that match that pattern in the same ZIP Code, allowing you to correct a partial or inaccurate address record that would otherwise result in an undeliverable mail piece.

The records that Street Search returns describe ranges of delivery address, not necessarily specific individual addresses. Therefore, Street Search cannot be used to construct address records from partial data. It can be used to verify that a submitted address falls within a valid range of known addresses (and thus is *probably* a deliverable address). Alternately, it can used to suggest alternate spellings for a street name or possibly alternate ZIP codes within the same city when the Address Verifier service cannot verify a submitted address.

Street Search Options

The Web Smart Street Search service has two optional settings which control the extent of its search.

Street In Range Search

This option will search for street data records within the submitted ZIP Code, that match the submitted street name and where the submitted street number falls within the primary range values.

For example, if the submitted address is “1234 Main St,” in the ZIP Code 12345, assuming there is a Main St in that ZIP Code, Street Search will return any street data records in that ZIP Code that match the name “Main St” and where the submitted street number falls within the primary range values. Therefore, if the low range was 1200 and the high was 1299, and the Odd or Even indicator was either “Even” or “Both,” then the record would be a match.

Street Name Search

This option will search for street data records within the submitted ZIP Code, that match the submitted street name.

Using the above example, Street Search would return any record that matched “Main St.” Note that Street Search only searches on the street name, independent of the suffix, so if there was a “Main Ave” and a “Main Blvd” in the same ZIP Code, Street Search would return those records as well.

Understanding Street Search

If Web Smart Street Search cannot find an exact match using the submitted data, it expands the search using the following logic.

Partial Matches

If Street Search does not find a match for the submitted street name, it will append a wildcard character (“*”) to the submitted name and search for any records where the beginning of the street name matches this pattern. If it still

does not find a match, Street Search will shorten the search string by one character and search again, continuing this process until it either finds matching records or until it is search on only the first character of the street name.

ZIP Code

Street Search will use the submitted ZIP Code, if present. If no valid ZIP Code is present, then Street Search will use the submitted city and state. If no valid ZIP Code, city or state are submitted, Street Search will return an error.

If a search of the submitted ZIP Code does not produce a match, Street Search will expand the search to within the submitted city/state combination, if any.

Adding Street Search to a Project

If you are using the SOAP service with Visual Studio .NET, you need to add a web reference to the service to your project. Click on the **Project** menu and select *Add Web Reference...* Enter the following URL on the Add Web Reference dialog box:

```
https://streetsearch.melissadata.net/v2/SOAP/Service.svc
```

If you are not using Visual Studio .NET, see the documention for your SOAP interface for the procedure for adding the service to your project.

Submitting an XML Request

After building your XML string from your data, an XML request to the web service is submitted using an HTTP POST operation to the following URL:

```
https://streetsearch.melissadata.net/v2/XML/Service.svc/
   doStreetSearch
```

Building a REST Request

Query strings are sent to the webservice as part of the URL using an HTTP Get operation appended to following URL:

```
https://streetsearch.melissadata.net/v2/REST/Service.svc/
   doStreetSearch
```

3 Street Search Request Object

At the minimum, a request to the WebSmart Street Search service will consist of a street address and either a ZIP Code or a combination of city and state. Only one record is allowed per request.

SOAP Request

The following Visual Basic Code shows a simple order of operations for building and submitting a Request object, submitting it to the Web Service and retrieving a response object.

Step 1 – Create the Request and Response Objects

```
Dim ReqStreetSearch As New dqwsStreetSearch.Request
Dim ResStreetSearch As New dqwsStreetSearch.Response
```

Step 2 – Assign the General Request Values

There are three properties of the Request object that apply to the request as a whole. CustomerID is required.

```
ReqStreetSearch.CustomerID = strCustID
ReqStreetSearch.TransmissionReference = strTranRef
ReqStreetSearch.OptInRangeOnly = True
```

The Transmission Reference is a unique string value that identifies this request.

Step 3 – Build the Record

The exact method for building the array will depend on the exact database software in use, but you will need to assign the required values to the corresponding elements in the Request.

```
ReqStreetSearch.Address = "22382 Avenida Empresa"
ReqStreetSearch.Zip = "92688"
ReqStreetSearch.City = "Rancho Santa Margarita"
ReqStreetSearch.State = "CA"
```

Step 4 – Submit the Request

The final step is to create the Service Client Object and then submit the Request object doStreetSearch method. This sends the data to the web service and retrieves the Response object.

```
StreetSearchClient = New dqwsStreetSearch.Service
ResStreetSearch =
   StreetSearchClient.doStreetSearch(ReqStreetSearch)
```

XML Request

The raw XML request is built using whatever XML tools are available via your development tools and submitted to the following URL using an HTTP POST request:

```
https://streetsearch.melissadata.net/v2/XML/Service.svc/
   doStreetSearch
```

The following XML Code contains the same request as the SOAP example above.

```
<Request>
   <TransmissionReference>Web Service Test 2008/12/31
   </TransmissionReference>
   <CustomerID>123456789</CustomerID>
   <OptInRangeOnly>True</OptInRangeOnly>
   <AddressLine>22382 Avenida Empresa</AddressLine>
   <City>Rancho Santa Margarita</City>
   <State>CA</State>
   <Zip>92688</Zip>
   <Country />
</Request>
```

REST Request

A REST request can submit a single address record via an HTTP GET. The following example uses the same address as the SOAP and XML samples.

```
https://streetsearch.melissadata.net/v2/REST/Service.svc/
   doStreetSearch?id=12345678&opt=true&a=22382%20Avenida%
   20Empresa&city=Rancho%20Santa%20Margarita&state=CA&zip
   =92688
```

Request Elements

The following section lists the elements that set the basic options for each and identify the user to the Web Service.

Customer ID

This is a required string value containing the identifier number issued to the customer when signing up for Melissa Data Web Services.

Remarks

You need a customer ID to access any Melissa Data Web Service. If this element is not populated, the web service will return an error. To receive a customer ID, call your Melissa Data sale representative at 1-800-MELISSA.

Syntax

SOAP

```
Request.CustomerID = string
```

XML

```
<Request>
   <CustomerID>String</CustomerID>
</Request>
```

REST

```
id={CustomerID}
```

Transmission Reference

This is an optional string value that may be passed with each Request to serve as a unique identifier for this set of records.

Remarks

This value is returned as sent by the Response Array, allowing you to match the Response to the Request.

Syntax

SOAP

```
Request.TransmissionReference = string
```

XML

```
<Request>
    <TransmissionReference>String</TransmissionReference>
</Request>
```

REST

```
t={transMissionReference}
```

OptInRangeOnly

This element enables the Street In Range search option of the WebSmart Street Search service.

Remarks

Setting this option to “True” will cause StreetSearch to search for street data records within the submitted ZIP Code, that match the submitted street name and where the submitted street number falls within the primary range values.

For example, if the submitted address is “1234 Main St,” in the ZIP Code 12345, assuming there is a Main St in that ZIP Code, Street Search will return any street data records in that ZIP Code that match the name “Main St” and where the submitted street number falls within the primary range values. Therefore, if the low range was 1200 and the high was 1299, and the Odd or Even indicator was either “Even” or “Both,” then the record would be a match.

If this element is set to “False” or left blank, StreetSearch will search for street data records within the submitted ZIP Code, that match the submitted street name.

Using the above example, Street Search would return any record that matched “Main St.” Note that Street Search only searches on the street name, independent of the suffix, so if there was a “Main Ave” and a “Main Blvd” in the same ZIP Code, Street Search would return those records as well.

Syntax

SOAP

```
Request.OptInRangeOnly = string
```

XML

```
<Request>
   <OptInRangeOnly>String</OptInRangeOnly>
</Request>
```

REST

```
opt={OptInRangeOnly}
```

AddressLine

This element passes the street address to the Street Search service.

Remarks

The AddressLine element is required with any request to the Street Search service. Attempting to submit a request without a populated AddressLine element will result in an error.

AddressLine must contain, at the very minimum, a street number and a full or partial street name. For example, “1234 M” would match all street names beginning with the letter M in the submitted ZIP Code.

Syntax

SOAP

```
Request.AddressLine = string
```

XML

```
<Request>
    <AddressLine>String</AddressLine>
</Request>
```

REST

```
a={AddressLine}
```

City

This element passes the city name to the Street Search service.

Remarks

The city element is optional, as long as the ZIP Code is submitted with the request. However, if the ZIP Code is missing or not valid, Street Search will fall back on the city and state to determine the ZIP Code.

Also, if a search of the primary ZIP Code returns no results, Street Search will search the city and state combination for other ZIP codes and use those ZIP codes to expand the search.

Syntax

SOAP

```
Request.City = string
```

XML

```
<Request>
    <City>String</City>
</Request>
```

REST

```
city={City}
```

State

This element passes the state abbreviation to the Street Search service.

Remarks

The state element is optional, as long as the ZIP Code is submitted with the request. However, if the ZIP Code is missing or not valid, Street Search will fall back on the city and state to determine the ZIP Code.

Also, if a search of the primary ZIP Code returns no results, Street Search will search the city and state combination for other ZIP codes and use those ZIP codes to expand the search.

Syntax

SOAP

```
Request.State = string
```

XML

```
<Request>
    <State>String</State>
</Request>
```

REST

```
state={State}
```

Zip

This element passes a five-digit ZIP Code to the Street Search service.

Remarks

The ZIP element is required but, if missing or not valid, Street Search will fall back on the city and state to determine the ZIP Code.

Also, if a search of the primary ZIP Code returns no results, Street Search will search the city and state combination for other ZIP codes and use those ZIP codes to expand the search.

Syntax

SOAP

```
Request.Zip = string
```

XML

```
<Request>
   <Zip>String</Zip>
</Request>
```

REST

```
zip={Zip}
```

Country

This element passes the two-characer country code to the Street Search service.

Remarks

Currently, Street Search only works on addresses within the United States, so this element is optional and has no effect. It is included for future compatibility.

Syntax

SOAP

```
Request.Country = string
```

XML

```
<Request>
    <Country>String</Country>
</Request>
```

REST

```
ctry={Country}
```

4 Street Search Response Object

Combining Street Record components into a full address line

To display to the end user, you may want to combine the separate address elements of the street record into a full address. For your typical address line, the order you would combine the address components would be:

```
[Primary Range] [Pre-Direction] [Street Name] [Suffix]
[Post-Direction] [Suite Name] [Suite Range]
```

Not all addresses will have all of these components, in which case they will be blank. However, there are exceptions. If the street record is a PO Box record or a Rural Route record, you will have to re-arrange the address components differently.

PO Boxes (Address Type P):

For PO Boxes, “PO Box” is stored in the street name while the box number is stored in the primary range. There are no directionals, suffixes, or suites for PO Boxes.

```
[Street Name] [Primary Range]
```

Rural Routes (Address Type R):

For Rural Routes, the street name contains the route information. Additionally, if the route has multiple mailboxes, the primary range will have the box range information. USPS specifications do not have a place to put the word 'Box' for these types of addresses, so you must add it is manually if a box primary

range exists. Like PO Boxes, Rural Routes do not have directionals, suffix, or suites.

```
[Street Name] Box [Primary Range]
```

Spanish Addresses:

Suffixes are usually positioned after the street name. However, Spanish style streets have their suffixes before the street name. If the suffix is “Avenida;” “Calle;” “Camino;” “Paseo;” or “Via,” position the address components as such:

```
[Primary Range] [Pre-Direction] [Suffix] [Street Name]
[Post-Direction]
```

All of these components are returned by each street record, except for the primary range which comes in the form of a number range with a primary range low and a primary range high. When doing an OptInRangeOnly search, you will assume the provided input range is correct, and use that for display. To do so, make sure to keep the parsed primary range from Address Object or the WebSmart AddressCheck Service.

Non-Deliverable Street Records

Sometimes an entire range on a street will be non-deliverable. In those cases, the street record still exists but the Plus4High and the Plus4High will both contain the string “XXXX”. It is recommended that you filter out these records as they cannot contain deliverable addresses.

Using StreetSearch with AddressCheck Error Result Codes

StreetSearch can be combined with the results of an AddressCheck to find alternatives for addresses that have uncorrectable errors. Of high importance is the use of OptInRangeOnly, which should be turned on if the input primary range was not the source of the address error. Here are types of AddressCheck Result errors and how to use StreetSearch for each of them.

AE01 - Zip Code Error

Unfortunately, if the zip code and city/state are missing or incorrect, we cannot search for alternative addresses.

AE02 - Unknown Street

The input street was not found, so we should assume the input range is correct and turn on OptInRangeOnly to search for streets for which the input range is valid.

AE03 - Component Mismatch Error

This means the street name and range matched but other multiple address components did not match. Turn on OptInRangeOnly to find the correct street record and display the correct components.

AE04 - Non-deliverable Address

The physical address is correct but no mailbox exist there. Since the address is correct, there are no alternative to generate.

AE05 - Multiple Match

Most commonly, this is due to multiple suffixes for the same street name. Turn on OptInRangeOnly to isolate the possible records.

AE06 - Early Warning Address

Address does not exist in current database but the street is scheduled to be added to the next update. No alternative generation is recommended here.

AE07 - Empty Address Input

There was no data to generate an alternative for.

AE08, AE09 - Suite Range Error, Suite Range Missing

This means the primary range and street name were correct. Turn on OptInRangeOnly to isolate the records in the highrise. Additionally, USPS data often has one or two records without suites in the beginning to represent the entire building and/or building office. To list possible suites, exclude those records by making sure a street record has a suite range low and suite range high.

AE10, AE11 - Primary Range Invalid, Primary Range Missing

Here, the primary range is the incorrect piece of information, so we should **not** turn on OptInRangeOnly. Doing a Street search with OptInRangeOnly off will return all possible ranges for the input street.

AE12, AE13 - PO Box or Rural Route Number is invalid, missing

Here, the PO Box and Rural Route number is equivalent to the primary range, so we should NOT turn on OptInRangeOnly. Doing a Street search with OptInRangeOnly off will return all possible ranges for the input PO Box or Rural Route.

AE14 - CMRA private mailbox missing

The USPS does not possess data for CMRA (Commercial Mail Receiving Agencies) so alternatives cannot be generated.

Return Elements — General

The SOAP interface for the Street Search service returns a Response Object. The primary component of this object is an array of StreetRecord objects, one for each street data record returned.

The XML and REST interfaces return XML documents containing a number of <StreetRecord> elements, one for each street record returned.

TransmissionReference

Returns a string value containing the contents of the TransmissionReference element from the original Request.

Remarks

If you passed any value to the TransmissionReference element when building your request, it is returned here. You can use this property to match the response to the request.

Syntax

SOAP

```
string = Response.TransmissionReference
```

XML

```
<ResponseArray>
   <TransmissionReference>
      String
   </TransmissionReference>
</ResponseArray>
```

Total Records

Returns a string value containing the number of street records returned with the current response.

Remarks

This property returns the number of street records processed and returned by the response as a string value.

Syntax

SOAP

```
string = Response.TotalRecords
```

XML

```
<ResponseArray>
   <TotalRecords>String</TotalRecords>
</ResponseArray>
```

Results

Returns a string value containing the general error, system error, and search error messages from the most recent request sent to the service.

Remarks

The following codes can be returned:

Code	Short Description	Long Description
SE01	Web Service Internal Error	The web service experienced an internal error.
GE01	Empty Request Structure	The SOAP, JSON, or XML request structure is empty.
GE02	Empty Request Record Structure	The SOAP, JSON, or XML request record structure is empty.
GE03	Records Per Request Exceeded	The counted records sent more than the number of records allowed per request.
GE04	Empty CustomerID	The CustomerID is empty.
GE05	Invalid CustomerID	The CustomerID is invalid.
GE06	Disabled CustomerID	The CustomerID is disabled.
GE07	Invalid Request	The SOAP, JSON, or XML request is invalid.
SS01	Records Within Range Returned	The street search returned one or more results within the designated range (OptInRangeOnly set to true).
SS02	Records With Street Name Returned	The street search returned one or more results with the street name only (OptInRangeOnly set to false).
SS03	First 1000 Streets Returned	1,000 street records were returned, but more than 1,000 were found. Please narrow your search.
DE01	Missing or Invalid Input	The required input for the service was not met or was in an invalid format.
DE02	No Records Found	There was a search error: No records were found.

Syntax

SOAP

```
string = Response.Results
```

XML

```
<ResponseArray>
   <Results>String</Results>
</ResponseArray>
```

Version

Returns a string value containing the current version number of the Street Search service.

Syntax

SOAP

```
string = Response.Version
```

XML

```
<ResponseArray>
   <Version>String</Version>
</ResponseArray>
```

StreetRecord Elements

The SOAP version of the Response object returns a property called StreetRecord which is an array of StreetRecord objects matching the input submitted with the Request with the original .

The XML and REST services may return one or more <StreetRecord> elements that match the original request.

The maximum number of records returned in both cases in 1,000.

The following section describes the elements returned by each record in the Repsonse Object.

Record ID

For each record in the Response, this element returns a string value containing the sequential identifier for the current StreetRecord.

Remarks

The RecordID is sequential number to aid in organizing the StreetRecords returned by the Street Search service.

Syntax

SOAP

```
string = Response.StreetRecord().RecordID
```

XML and REST

```
<Response>
   <StreetRecord>
      <RecordID>String</RecordID>
   </StreetRecord>
</Response>
```

Company

For each record in the StreetRecord array, this element returns the string value containing the contents of the Company element if a company is tied to a street record in the USPS Data.

Syntax

SOAP

```
string = Response.StreetRecord().Company
```

XML

```
<Response>
   <StreetRecord>
      <Company>String</Company>
   </StreetRecord>
</Response>
```

FullAddressLine

For each record in the StreetRecord array, this element returns a string value containing the complete address line.

Remarks

This property returns the range high and low for both the street and the suite, as well as an indicator if it is odd or even.

Example

```
"22342-22342 Even AVENIDA EMPRESA STE 225-225 Odd"
```

Syntax

SOAP

```
string = Response.StreetRecord().FullAddressLine
```

XML

```
<Response>
   <StreetRecord>
      <FullAddressLine>String</FullAddressLine>
   </StreetRecord>
</Response>
```

PrimaryRange

For each record in the StreetRecord array, the PrimaryRange node returns three elements that indicate the high and low values of the range of street numbers, and indicates whether the numbers in the range are even, odd, or both.

Remarks

The high and low elements return a 10-character maximum string containing the first and last street numbers in the range of streets address represented by the current StreetRecord. For example, if the address range is 100 to 200 Main Street, these elements will return the values “100” and “200,” respectively.

A hyphen in front of the range element indicates a significant leading zero. That means that the leading zero is part of the range and is required. For example, -7 would indicate the range is 07 and cannot be just 7.

The OddEven element returns a “E” if the numbers in the range are all even, “O” if they are all odd, and “B” if there are both odd and even.

Syntax

SOAP

```
string = Response.StreetRecord().PrimaryRange.Low
string = Response.StreetRecord().PrimaryRange.High
string = Response.StreetRecord().PrimaryRange.OddEven
```

XML

```
<Response>
   <StreetRecord>
      <PrimaryRange>
         <Low>String</Low>
         <High>String</High>
         <OddEven>String</OddEven>
      </PrimaryRange>
   </StreetRecord>
</Response>
```

Street

These elements return the parsed components of the street name: the pre-directional, the name, the suffix and the post-directional.

Remarks

The Street elements return the following information:

PreDirection

The pre-direction is a geographical directional that precedes the street name. For example, in the address range of 100 N Main St, this element will hold the “N.” Directions such as “North” will be changed to “N” before they are returned by this function.

The directional can be one of the following: “N;” “NE;” “E;” “SE;” “S;” “SW;” “W;” “NW.”

Name

Returns the name of the street containing the current range.

Suffix

When street names are used more than once within a city or area, street suffixes are often used to distinguish them. For example, if there is a Main Street but also a Main Avenue, this element will hold either the “St” or “Ave” suffix, depending on which one is being referred to.

Typical suffix values include “ST,” “RD,” “AVE,” “BLVD,” “CIR,” and “PL.”

PostDirection

The post-direction is a geographical directional that follows the street name. For example, in the address range of 100 N Main St E, this element will hold the “E.”

The post direction can be one of the following: “N;” “NE;” “E;” “SE;” “S;” “SW;” “W;” “NW.”

Note About Spanish Street Names

For street names that begin with certain Spanish words, that first word will be returned by the Suffix element and the next word in the street name will be returned as the street name. This is because Spanish street names have the suffix at the front of the address. If the suffix was not returned by the Suffix element, it would appear that every street in some Puerto Rican ZIP codes

began with the same letter. The Spanish words that would be returned by the Suffix element are: “Avenida;” “Calle;” “Camino;” “Paseo;” and “Via.”

Syntax

SOAP

```
string = Response.StreetRecord().Street.PreDirection
string = Response.StreetRecord().Street.Name
string = Response.StreetRecord().Street.Suffix
string = Response.StreetRecord().Street.PostDirection
```

XML

```
<Response>
   <StreetRecord>
      <Street>
         <PreDirection>String</PreDirection>
         <Name>String</Name>
         <Suffix>String</Suffix>
         <PostDirection>String</PostDirection>
      </Street>
   </StreetRecord>
</Response>
```

Suite

These elements return the secondary address information, if any, for the current Street record.

Remarks

The Suite elements return the following information:

Name

If suite numbers exist in the address range, this element will indicate the proper suite name for addresses in that range.

Possible return values are: “#;” “APT;” “BLDG;” “BOX;” “BSMT;” “DEPT;” “FL;” “FRNT;” “HNGR;” “LBBY;” “LOT;” “LOWR;” “OFC;” “PH” (Penthouse); “PIER;” “REAR;” “RM;” “SIDE;” “SLIP;” “SPC;” “STE;” “STOP;” “TRLR;” “UNIT;” “UPPR.

Low

This element returns the lowest suite number in the current range. For instance, if the range covered the suite numbers 1001 through 2001, this element would return “1001.”

High

This element returns the lowest suite number in the current range. For instance, if the range covered the suite numbers 1001 through 2001, this element would return “2001.”

OddEven

This element returns an “O” for Odd, an “E” for Even, or a “B” for Both. An “O” indicates that the suite range contains only odd numbers, an “E” indicates that only even numbers are present in the suite range, and a “B” indicates that both odd and even numbers are included in the suite range. For example, an “O” will indicate that, in the 1001 to 2001 suite range, only suite numbers 1001, 1003, 1005, 1007, and so on, exist.

Syntax

SOAP

```
string = Response.StreetRecord().Suite.Name
string = Response.StreetRecord().Suite.Low
string = Response.StreetRecord().Suite.High
string = Response.StreetRecord().Suite.OddEven
```

XML

```
<Response>
   <StreetRecord>
      <Suite>
         <Name>String</Name>
         <High>String</High>
         <Low>String</Low>
         <OddEven>String</OddEven>
      </Suite>
   </StreetRecord>
</Response>
```

Zip

The Zip elements return the five-digit ZIP Code, plus the high and low values of the four-digit ZIP + 4 extensions.

Remarks

The Zip5 element returns the five-digit base ZIP Code for the street record.

The return values of the Plus4Low and Plus4High elements define the range of ZIP + 4 codes for the street record. If the ZIP + 4 range is 1234 to 1334, Plus4Low will return the “1234” and Plus4High will return “1334.”

The Plus4Low and Plus4High elements will usually contain the same value. In some instances, such as with PO Boxes, the Plus4Low and Plus4High elements will return different values as every PO Box has its own ZIP + 4 code.

In these cases, the lengths of the address range and the Plus4 High-Low range will be identical and the numbers can be paired up accordingly. For example, if the address range is PO Box 100 to 200 and the Plus4 High-Low range is 0100 to 0200, “PO Box 110” will have a Plus4 of “0110.”

Syntax

SOAP

```
string = Response.StreetRecord().Zip.Zip5
string = Response.StreetRecord().Zip.Plus4Low
string = Response.StreetRecord().Zip.Plus4High
```

XML

```
<Response>
   <StreetRecord>
      <Zip>
         <Zip5>String</Zip5>
         <Plus4Low>String</Plus4Low>
         <Plus4High>String</Plus4High>
      </Zip>
   </StreetRecord>
</Response>
```

CarrierRoute

For each record in the Street Record Array, CarrierRoute returns a string value containing the four-character code defining the carrier route for that record.

Remarks

The first character of this property is always alphabetic, and the last three characters are numeric. The alphabetic letter indicates the type of delivery associated with this address.

B	PO Box
C	City Delivery
G	General Delivery
H	Highway Contract
R	Rural Route

Syntax

SOAP

```
string = Response.StreetRecord().CarrierRoute
```

XML

```
<Response>
   <StreetRecord>
      <CarrierRoute>String</CarrierRoute>
   </StreetRecord>
</Response>
```

Urbanization

For each record in the StreetRecord array, Urbanization Code returns a string value containing the six-digit code number for the Urbanization connected with the current address. Urbanization Name returns a string value containing the name of the Urbanization.

Remarks

Urbanizations will only be returned for addresses in Puerto Rico.

Syntax

SOAP

```
string = Response.StreetRecord().Urbanization.Code
string = Response.StreetRecord().Urbanization.Name
```

XML

```
<Response>
   <StreetRecord>
      <Urbanization>
         <Name>String</Name>
         <Code>String</Code>
      </Urbanization>
   </StreetRecord>
</Response>
```

AddressType

For each record in the StreetRecord array, this element returns the address type for the current streetrecord returned

Remarks

The Address Type function is a 1-character (maximum) string value, that indicates the type of address that was returned:

Code	Type
F	Firm or Company address
G	General Delivery address
H	High Rise or Business complex
P	PO Box address
R	Rural Route address
S	Street or Residential address

Syntax

SOAP

```
string = Response.StreetRecord().AddressType
```

XML

```
<Response>
   <StreetRecord>
      <AddressType>String</AddressType>
   </StreetRecord>
</Response>
```

LACSIndicator

For each record in the StreetRecord array, this element returns the LACS indicator for each StreetRecord.

Remarks

The LacsIndicator element returns a one-character which indicates whether or not the address returned by current StreetRecord element has undergone LACS conversion.

Some rural route addresses are modified to city-style addresses to allow emergency services (for example, ambulance, police, fire, and so on) to find these addresses more efficiently.

An empty space in the return value indicates that the address has not undergone a conversion. A value of “L” in the LACSIndicator return value indicates that the address has undergone a conversion. After a conversion, the old address is retained in the ZIP + 4 file for a period of one year. After the one year period, the old addresses will be dropped from the ZIP + 4 file and the address checking logic will not assign a ZIP + 4 for this address.

Syntax

SOAP

```
string = Response.StreetRecord().LACSIndicator
```

XML

```
<Response>
    <StreetRecord>
        <LACSIndicator>String</LACSIndicator>
    </StreetRecord>
</Response>
```

BaseAlternateIndicator

For each record in the StreetRecord array, this element returns the base alternate indicator for the current StreetRecord.

Remarks

This element returns a 1-character maximum string value which specifies whether or not a record is a base (preferred) or alternate record. Base records (indicated by a “B”) can represent a range of addresses or an individual address, such as a firm record, while alternate records (indicated by an “A”) are individual delivery points. Base records are generally preferred over alternate records. The base record for an alternate record can be found by matching up the ZIP + 4 ranges.

Syntax

SOAP

```
string = Response.StreetRecord().BaseAlternateIndicator
```

XML

```
<Response>
   <StreetRecord>
      <BaseAlternateIndicator>String</BaseAlternateIndicator>
   </StreetRecord>
</Response>
```

Response Object XML Format

The following shows the structure of the XML document returned by the WebSmart StreetSearch Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<Response>
   <Version>String</Version>
   <TransmissionReference>String</TransmissionReference>
   <Results>String</Results>
   <TotalRecords>String</TotalRecords>
   <StreetRecord>
      <RecordID>String</RecordID>
      <Company>String</Company>
      <FullAddressLine>String</FullAddressLine>
      <PrimaryRange>
         <Low>String</Low>
         <High>String</High>
         <OddEven>String</OddEven>
      </PrimaryRange>
      <Street>
         <PreDirection>String</PreDirection>
         <Name>String</Name>
         <Suffix>String</Suffix>
         <PostDirection>String</PostDirection>
      </Street>
      <Suite>
         <Name>String</Name>
         <Low>String</Low>
         <High>String</High>
         <OddEven>String</OddEven>
      </Suite>
      <Zip>
         <Zip5>String</Zip5>
         <Plus4Low>String</Plus4Low>
         <Plus4High>String</Plus4High>
      </Zip>
      <CarrierRoute>String</CarrierRoute>
      <Urbanization>
         <Code>String</Code>
         <Name>String</Name>
```

```
            </Urbanization>
            <AddressType>String</AddressType>
            <LACSIndicator>String</LACSIndicator>
            <BaseAlternateIndicator>
               String
            </BaseAlternateIndicator>
         </StreetRecord>
      </Response>
```