Copyright

The information furnished in this document and software was published by Intelligent Search Technology. Intelligent Search Technology holds a non-exclusive license to publish and sell ZIP+4, e-LOT, and ZIPMove information. The price of CorrectAddress® is neither established, nor controlled or approved by the United States Postal Service. ZIP+4, e-LOT, CASS, DPV, LACS LE®, Suite Link™ and ZIPMove are registered trademarks of the United States Postal Service. Any advertising of this product was neither approved nor endorsed by the United States Postal Service.

© United States Postal Service 2010

Trademarks

NameSearch® and CorrectAddress® are registered trademarks of Intelligent Search Technology, Ltd.

Microsoft, Visual Studio, Access. Microsoft.NET Framework and Windows are registered trademarks of the Microsoft Corporation in the United States and other countries.

Other product names mentioned in this manual may be a trademark or trademarks of their respective companies and are hereby acknowledged.
Table of Contents

CHAPTER 1 - GENERAL INFORMATION.............................................................................. 1-1
  Product Information ...................................................................................................... 1-1
  Web Service and Object Model Information .............................................................. 1-1
  What’s New ..................................................................................................................... 1-3
  web Service Location (URL) ............................................................................................ 1-4
  Main Functions .............................................................................................................. 1-4
  Account Management Functions .................................................................................. 1-5
  Output Data Definitions ............................................................................................... 1-6

CHAPTER 2 - FUNCTION SPECIFICATIONS ..................................................................... 2-1
  wsCorrectA .................................................................................................................... 2-1
  wsTigerCA ...................................................................................................................... 2-4
  wsFindCityCounty ....................................................................................................... 2-5
  wsFindZipCity ............................................................................................................... 2-6
  wsFindCityState .......................................................................................................... 2-7
  wsStreetSearch2 .......................................................................................................... 2-8
  wsGetAccountAccess ................................................................................................... 2-10
  wsGetAccountInfo ....................................................................................................... 2-11
  wsCreateBatch ............................................................................................................ 2-12
  wsDeleteBatch ............................................................................................................. 2-13
  wsGetBatchList ........................................................................................................... 2-14
  wsGetPS3553form ....................................................................................................... 2-15

CHAPTER 3 - WEB SERVICE INTEGRATION .................................................................... 3-1
  C# ................................................................................................................................. 3-1
  ColdFusion ................................................................................................................... 3-2
  Java ............................................................................................................................... 3-3
  Oracle ........................................................................................................................... 3-4
  PHP ............................................................................................................................... 3-5
  VB.NET ......................................................................................................................... 3-6
  CLASSIC ASP ............................................................................................................... 3-7

APPENDIX A - TRANSITIONING TO THE NEW OBJECT MODEL .................................. A-1
Chapter 1 - General Information

PRODUCT INFORMATION

The CorrectAddress® software engine is CASS™-certified by the United States Postal Service and SERP-certified by Canada Post to perform address correction, standardization and validation. The product is designed to overcome wide variations in address data, fix misspellings and erroneous information, fill in omitted components and normalize incorrect formatting. The output contains postal data records, in which components are separated into individual fields.

WEB SERVICE AND OBJECT MODEL INFORMATION

Web services allow client-side programs to invoke certain methods exposed by a server and receive any processed data back. Web services provide a standard means of interaction between different software applications, running on a variety of platforms and/or frameworks.

Communication between the client and server modules is done via Extensible Markup Language (XML) that allows for efficient data parsing in web-enabled applications. Most application frameworks provide predefined modules that aid with parsing XML documents. Furthermore, popular frameworks provide a mechanism through which the XML communication is entirely transparent to the developer – s/he simply deals with ‘objects’. The underlying framework translates the objects to an XML stream (known as serialization). Similarly, the data returned back from the server is translated from XML to more easily manageable objects (known as deserialization) by the framework.

Before we take a look at exactly how one would invoke a web service, a few terms need to be understood.

**eXtensible Markup Language (XML):** Language to describe data and control flow in a generic, platform independent fashion

**Web Service Description Language (WSDL):** XML based language which provides a model for describing web services. The server uses WSDL to provide the following information:

1. Enumerate the various services/methods it offers
2. Name and type of arguments that these methods operate upon
3. Name and type of values (if any) returned by the server as a result of calling these methods

**Simple Object Access Protocol (SOAP):** Once you know what services are offered by the server, SOAP is a means through which the client communicates with the server to actually invoke those services

**Serialization:** The process of converting an object or a data type (e.g., int, string, etc.) into a generic XML equivalent

**Deserialization:** The process of converting an XML representation of an object or data type into a more framework specific form

In order to call a web service, one must:

1. Obtain the URL to the WSDL
2. Point the application framework to this URL – the framework will create stubs that handle communication with the server
3. Write the client program that uses the stubs generated above to call the web services

The above steps will be evident in the demo programs described in Chapter 3: Web Service Integration later in this manual.
The following figure depicts a typical client/server interaction via web services:

1. The Client calls web service methods like any other regular method, ‘thinking’ that the client stub offers these methods.
2. The Client stub serializes the method arguments and creates a SOAP request that it sends over to the server.
3. The server receives the SOAP request, deserializes it and calls the requested method with the arguments provided by the client.
4. The server serializes the return values from the methods into a SOAP response that it sends back to the client.
5. The client stub receives the SOAP response and deserializes it into a more useful object.
6. The returned object is then sent over to the Client module.
WHAT’S NEW

- In November 2009, Intelligent Search Technology introduced a new set of web services utilizing an enhanced XML object model. Among the changes are:
  - Improved speed
  - Improved code efficiency
  - Improved programmer’s interface – no more parsing XML responses. The new web services return result objects that have the information parsed and ready for use
  - Code has been refactored, leading to deprecation of some outdated services and packing additional functionality into existing methods

If you need help transitioning from the old-style web services to the new object model, please refer to Appendix A, or contact IST Technical Support Team.
WEB SERVICE LOCATION (URL)

https://www.intelligentsearch.com/CorrectAddressWS/CorrectAddressWebService.asmx

MAIN FUNCTIONS

The following is a list of currently supported CorrectAddress® web service functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsCorrectA</td>
<td>Validates and CASS-standardizes input address with Delivery Point Validation (DPV) and LACSLink™ processing. Verifies and corrects Canadian addresses. All near matches are returned when applicable. Number of remaining transactions is returned in the SearchesLeft output field.</td>
</tr>
<tr>
<td>wsTigerCA</td>
<td>Validates, CASS-standardizes and geo-codes input address with Delivery Point Validation (DPV) and LACSLink™ processing. All near matches are returned when applicable. Number of remaining transactions is returned in the SearchesLeft output field.</td>
</tr>
<tr>
<td>wsFindCityCounty</td>
<td>Accepts ZIP code as input. Returns preferred city name, state, county information. Number of remaining transactions is returned in the SearchesLeft output field.</td>
</tr>
<tr>
<td>wsFindZipCity</td>
<td>Accepts city and state as input. Returns all corresponding ZIP codes. Number of remaining transactions is returned in the SearchesLeft output field.</td>
</tr>
<tr>
<td>wsFindCityState</td>
<td>Accepts ZIP code as input. Returns all valid mailing city names, state, county information. Number of remaining transactions is returned in the SearchesLeft output field.</td>
</tr>
<tr>
<td>wsStreetSearch2</td>
<td>Accepts street information with city or ZIP code Returns matches based on similar street records in the area. The matches are unsorted and contain street number ranges and secondary number ranges (if applicable). Number of remaining transactions is returned in the SearchesLeft output field.</td>
</tr>
</tbody>
</table>
## ACCOUNT MANAGEMENT FUNCTIONS

The following functions are used to access or modify user account information:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsGetAccountAccess</td>
<td>Returns information about account access privileges.</td>
</tr>
<tr>
<td>wsGetAccountInfo</td>
<td>Returns account related information. Currently, it is used to retrieve remaining number of transactions. Return codes are: 0 – information retrieved successfully 1 – invalid account 2 – account is disabled 3 – account does not have access to CorrectAddress® XML web services 4 – unspecified error</td>
</tr>
<tr>
<td>wsCreateBatch</td>
<td>Creates user batch space. The batch space stores all settings necessary to produce a USPS Postal Form 3553 (CASS report) upon completion of a batch job. The following functions allow users to append current transaction to a specific batch space by passing its name in the “batch name” parameter: wsCorrectA, wsTigerCA. Note: The batch space only stores job statistics, <em>not</em> the actual output address data. Storing address data returned by web services is users’ responsibility.</td>
</tr>
<tr>
<td>wsDeleteBatch</td>
<td>Empties and removes specified user batch space.</td>
</tr>
<tr>
<td>wsGetBatchList</td>
<td>Returns all current batch spaces under user’s account.</td>
</tr>
<tr>
<td>wsGetPS3553form</td>
<td>Retrieves current batch job statistics necessary to generate USPS Postal Form 3553 (CASS report). Data is returned in the XML format.</td>
</tr>
</tbody>
</table>
## OUTPUT DATA DEFINITIONS

The following data fields are contained in the web service response string. Keyword “Any” is used to designate fields that may contain any valid ASCII characters. Note that numeric fields may contain leading zeros.

<table>
<thead>
<tr>
<th>OUTPUT DATA</th>
<th>FIELD TYPE</th>
<th>MAXIMUM FIELD SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreetNumber</td>
<td>Alphanumeric, Fractions</td>
<td>10 (21 for wsStreetSearch2 service)</td>
</tr>
<tr>
<td>PreDirectional</td>
<td>Alpha</td>
<td>2</td>
</tr>
<tr>
<td>StreetName</td>
<td>Alphanumeric</td>
<td>28</td>
</tr>
<tr>
<td>StreetSuffix</td>
<td>Alpha</td>
<td>4</td>
</tr>
<tr>
<td>PostDirectional</td>
<td>Alpha</td>
<td>2</td>
</tr>
<tr>
<td>SecondaryDesignation</td>
<td>Alpha, #</td>
<td>4</td>
</tr>
<tr>
<td>SecondaryNumber</td>
<td>Alphanumeric, Fractions</td>
<td>8 (17 for wsStreetSearch2 service)</td>
</tr>
<tr>
<td>City/Municipality(US/Canada)</td>
<td>Alphanumeric</td>
<td>28</td>
</tr>
<tr>
<td>State/Province (US/Canada)</td>
<td>Alpha</td>
<td>2</td>
</tr>
<tr>
<td>ZipAddon/Postal Code</td>
<td>Numeric, -</td>
<td>10</td>
</tr>
<tr>
<td>Zip/Postal Code</td>
<td>Numeric</td>
<td>5</td>
</tr>
<tr>
<td>Addon</td>
<td>Numeric</td>
<td>4</td>
</tr>
<tr>
<td>LOTNumber</td>
<td>Numeric</td>
<td>5</td>
</tr>
<tr>
<td>DPCCheckdigit</td>
<td>Numeric</td>
<td>3</td>
</tr>
<tr>
<td>RecordType</td>
<td>Numeric</td>
<td>1</td>
</tr>
<tr>
<td>DeliveryLine1</td>
<td>Any</td>
<td>64</td>
</tr>
<tr>
<td>DeliveryLine2</td>
<td>Any</td>
<td>64</td>
</tr>
<tr>
<td>LastLine</td>
<td>Alphanumeric</td>
<td>64</td>
</tr>
<tr>
<td>LACS</td>
<td>Alpha</td>
<td>1</td>
</tr>
<tr>
<td>CarrierRoute</td>
<td>Alphanumeric</td>
<td>4</td>
</tr>
<tr>
<td>PMBDesignator</td>
<td>Alphanumeric, #</td>
<td>12</td>
</tr>
<tr>
<td>FirmRecipient</td>
<td>Any</td>
<td>40</td>
</tr>
<tr>
<td>Urbanization</td>
<td>Alpha</td>
<td>28</td>
</tr>
<tr>
<td>CountyName</td>
<td>Alpha</td>
<td>25</td>
</tr>
<tr>
<td>OUTPUT DATA</td>
<td>FIELD TYPE</td>
<td>MAXIMUM FIELD SIZE</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>CountyNumber</td>
<td>Numeric</td>
<td>3</td>
</tr>
<tr>
<td>Filler&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Alphanumeric</td>
<td>260 (see sample output for wsCorrectA function call)</td>
</tr>
<tr>
<td>GeoTLID (TLID)&lt;sup&gt;3&lt;/sup&gt;</td>
<td>Numeric</td>
<td>10 * 20</td>
</tr>
<tr>
<td>(Tigerstcode)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Numeric</td>
<td>2</td>
</tr>
<tr>
<td>(Tigercroute)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Alphanumeric</td>
<td>4</td>
</tr>
<tr>
<td>(Tigercounty)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Numeric</td>
<td>3</td>
</tr>
<tr>
<td>GeoMatchFlag (MatchFlag)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Alpha</td>
<td>1 * 20</td>
</tr>
<tr>
<td>GeoTract (Tract)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Numeric</td>
<td>6 * 20</td>
</tr>
<tr>
<td>GeoBlock (Block)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Numeric</td>
<td>4 * 20</td>
</tr>
<tr>
<td>GeoFLat (fLat)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Decimal with + sign</td>
<td>11 * 20</td>
</tr>
<tr>
<td>GeotLat (tLat)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Decimal with + sign</td>
<td>11 * 20</td>
</tr>
<tr>
<td>GeoFLong (fLong)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Decimal with – sign</td>
<td>12 * 20</td>
</tr>
<tr>
<td>GeotLong (tLong)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Decimal with – sign</td>
<td>12 * 20</td>
</tr>
<tr>
<td>GeoAddonStart(AddonStart)&lt;sup&gt;3&lt;/sup&gt;</td>
<td>Numeric</td>
<td>4 * 20</td>
</tr>
<tr>
<td>GeoAddonEnd(AddonEnd)&lt;sup&gt;3&lt;/sup&gt;</td>
<td>Numeric</td>
<td>4 * 20</td>
</tr>
<tr>
<td>GeoErrCodes (Tigererrcode)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Numeric</td>
<td>30</td>
</tr>
<tr>
<td>GeoRetCode (tigerRet)&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Numeric</td>
<td>10</td>
</tr>
<tr>
<td>ReturnCodes</td>
<td>Numeric</td>
<td>10</td>
</tr>
<tr>
<td>ErrorCodes&lt;sup&gt;3&lt;/sup&gt;</td>
<td>Alphanumeric</td>
<td>30</td>
</tr>
<tr>
<td>ErrorDesc&lt;sup&gt;4&lt;/sup&gt;</td>
<td>Alphanumeric</td>
<td>n/a</td>
</tr>
<tr>
<td>SearchesLeft&lt;sup&gt;5&lt;/sup&gt;</td>
<td>Numeric</td>
<td>n/a</td>
</tr>
</tbody>
</table>

<sup>1</sup> Output field for geo-coding only. The name in parentheses is used by TigerCA function call only.

<sup>2</sup> Output field for wsCorrectA and wsTigerCA function calls only.

<sup>3</sup> ErrorCodes
This node lists all codes relevant to the input address. In case of an incorrect username/password combination ErrorCodes is set to xx. ErrorCodes also equals xx when an account has no remaining transactions or has been deactivated.

<sup>4</sup> ErrorDesc
This node lists detailed messages for each error code in ErrorCodes.

<sup>5</sup> SearchesLeft
This node returns the number of transactions remaining on user account.
Please note that when `wsCorrectA` or `wsTigerCA` return multiple near-match records, only the first record will contain data for DeliveryLine1, DeliveryLine2, LastLine, LOTNumber, DPCCheckdigit, PMBDesignator, ReturnCodes, Error Codes, and ErrorDesc.

Complete list of CorrectAddress® fields, record types and error codes (U.S. and Canada) is available at:

Chapter 2 - Function Specifications

wsCorrectA

Description

Validates and CASS-standardizes input address with Delivery Point Validation (DPV) and LACSLink™ processing.

Verifies and corrects Canadian addresses.

All near matches are returned when applicable.

Number of remaining account transactions is returned in the SearchesLeft output field.

XML template

```xml
<username>string</username>
<password>string</password>
<firmname>string</firmname>
<urbanization>string</urbanization>
<delivery_line_1>string</delivery_line_1>
<delivery_line_2>string</delivery_line_2>
<city_state_zip>string</city_state_zip>
<ca_codes>string</ca_codes>
<ca_filler>string</ca_filler>
<batchname>string</batchname>
```

Parameters

- **username**: account username
- **password**: account password
- **firmname**: (Optional) firm or recipient name for input address
- **urbanization**: (Optional) urbanization name (Puerto Rico addresses only)
- **delivery_line_1**: input address line 1 information
- **delivery_line_2**: (Optional) input address line 2 information
- **city_state_zip**: input address city, state, ZIP information
  - (municipality, province, postal code for Canada)
- **ca_codes**: (Optional) additional codes for address parsing (see Remarks)
- **ca_filler**: (Optional) internal use (see Remarks)
- **batchname**: (Optional) name of batch space, if one exists, to which the specified address belongs to (this option is enabled for ability to produce ps3553 USPS forms).
Remarks

By default, output address is returned in capital letters. To enable mixed case output, pass "Mc" in the leftmost two available characters of ca_codes.

To enable street address parsing for no-match addresses, pass "Ry" in the leftmost two available characters of ca_codes.

DPV and LACSLink™ output data is returned through Filler output field (see sample output for wsCorrectA function call)

[Ex: CA_CODES = McRy ]

Output:

Output is an array of objects of type WsCorrectAddress:

```java
class WsCorrectAddress{
    string StreetNumber;
    string PreDirectional;
    string StreetName;
    string StreetSuffix;
    string PostDirectional;
    string SecondaryDesignation;
    string SecondaryNumber;
    string City;
    string State;
    string ZipAddon;
    string Zip;
    string Addon;
    string LOTNumber;
    string DPCCheckdigit;
    string RecordType;
    string DeliveryLine1;
    string DeliveryLine2;
    string LastLine;
    string LACS;
    string CarrierRoute;
    string PMBDesignator;
    string FirmRecipient;
    string Urbanization;
    string Urbanization;
    string CountyName;
    string CountyNumber;
    string Filler;
    string GeoTLID;
    string GeoMatchFlag;
    string GeoTract;
    string GeoBlock;
    string GeoLat;
    string GeoLat;
    string GeoLong;
    string GeoLong;
    string GeoAddEnd;
    string GeoAddStart;
    string GeoRetCode;
    string GeoErrCodes;
    string ReturnCodes;
    string ErrorCodes;
    string ErrorDesc;
    int SearchesLeft;
}
```
Example

Input:

```xml
<username>username</username>
<password>password</password>
<firmname>Intelligent Search Technology</firmname>
<urbanization>Urbanization</urbanization>
<delivery_line_1>445 Hamilton Ave 608</delivery_line_1>
<delivery_line_2></delivery_line_2>
<city_state_zip>White Plains, NY</city_state_zip>
<ca_codes>Mc</ca_codes>
<ca_filler></ca_filler>
<batchname></batchname>
```

The DPV and LACSLink™ output data is returned through *Filler* output field.

<table>
<thead>
<tr>
<th>Fill</th>
<th>Output Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Position (Index-1)</td>
<td>Length</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>255</td>
<td>1</td>
</tr>
<tr>
<td>256</td>
<td>2</td>
</tr>
</tbody>
</table>
wsTigerCA

Description

Validates, CASS-standardizes and geo-codes input address with Delivery Point Validation (DPV) and LACSLink™ processing.

All near matches are returned when applicable.

Number of remaining account transactions is returned in the SearchesLeft output field.

XML template

Same as in wsCorrectA Function.

Parameters

Same as in wsCorrectA Function.

Output:

Same as in wsCorrectA Function.

Example

Input:

```xml
<username>username</username>
<password>password</password>
<firmname>Intelligent Search Technology, Ltd</firmname>
<urbanization/>
<delivery_line_1>445 Hamilton Ave 608</delivery_line_1>
<delivery_line_2/></delivery_line_2>
<city_state_zip>White Plains, NY</city_state_zip>
<ca_codes>Mc</ca_codes>
<ca_filler/> </ca_filler>
<batchname/></batchname>
```
wsFindCityCounty

Description
Accepts ZIP code as input.
Returns city, state, county information.
Number of remaining account transactions is returned in the SearchLeft output field.

XML template

```xml
<username>string</username>
<password>string</password>
<zip>string</zip>
```

Parameters

username
account username
password
account password
zip
input ZIP code

Output:
Output is an object of type WsCityCounty:

```csharp
class WsCityCounty{
    string Zip;
    string City;
    string State;
    string CountyName;
    string CountyNum;
    int SearchLeft;
    string ErrorDesc;
}
```

Example

Input:

```xml
<username>username</username>
<password>password</password>
<zip>10509</zip>
```
**wsFindZipCity**

**Description**

Accepts city and state as input.

Returns all corresponding ZIP codes.

Number of remaining account transactions is returned in the `SearchLeft` output field.

**XML template**

```xml
<username>string</username>
<password>string</password>
<city>string</city>
<state>string</state>
```

**Parameters**

- `username`: account username
- `password`: account password
- `city`: input city name
- `state`: input state abbreviation

**Output:**

Output is an array of objects of type `WsZipCity`:

```csharp
class WsZipCity{
    string City;
    string State;
    string Zip;
    int SearchLeft;
    string ErrorDesc;
}
```

**Example**

**Input:**

```xml
<username>username</username>
<password>password</password>
<city>Ridgefield</city>
<state>CT</state>
```
**wsFindCityState**

**Description**

Accepts ZIP code as input.

Returns all valid mailing city names, state, county information.

Number of remaining account transactions is returned in the `SearchLeft` output field.

**XML template**

```
<username>string</username>
<password>string</password>
<zip>string</zip>
```

**Parameters**

- `username` account username
- `password` account password
- `zip` input ZIP code

**Output:**

Output is an array of objects of type WsCityState:

```
class WsCityState{
    string Zip;
    string City;
    string State;
    string CountyName;
    int SearchLeft;
    string ErrorDesc;
    }
```

**Example**

**Input:**

```
<username>username</username>
<password>password</password>
<zip>94707</zip>
```
wsStreetSearch2

Description

Accepts street information with city or ZIP code

Returns matches based on similar street records in the area. The matches are unsorted and contain street number ranges and secondary number ranges (if applicable). The ranges are returned in the following form:

Street number: START_RANGE (10 bytes) + END_RANGE (10 bytes) + OEB CODE (Odd, Even or Both, 1 byte)

Secondary number: START_RANGE (8 bytes) + END_RANGE (8 bytes) + OEB CODE (Odd, Even or Both, 1 byte)

Number of remaining account transactions is returned in the SearchLeft output field.

XML template

```xml
<username>string</username>
<password>string</password>
<firmname>string</firmname>
<urbanization>string</urbanization>
<delivery_line_1>string</delivery_line_1>
<delivery_line_2>string</delivery_line_2>
<city_state_zip>string</city_state_zip>
<mixedcase>boolean</mixedcase>
<brpathname>string</brpathname>
```

Parameters

- **username**: account username
- **password**: account password
- **firmname**: (Optional) firm or recipient name for input address
- **urbanization**: (Optional) urbanization name (Puerto Rico addresses only)
- **delivery_line_1**: input address line 1 information
- **delivery_line_2**: (Optional) input address line 2 information
- **city_state_zip**: input address city, state, ZIP information
- **mixedcase**: set to **True** for mixed case output, **False** for capitalized output
- **brpathname**: (Optional) name of batch space, if one exists, to which the specified address belongs to
  (this option is enabled for ability to produce ps3553 USPS forms)
Output:

Output is an array of objects of type WsStreetSearch:

```java
class WsStreetSearch{
    string StreetNumber;
    string PreDirectional;
    string StreetName;
    string StreetSuffix;
    string PostDirectional;
    string SecondaryDesignation;
    string SecondaryNumber;
    string City;
    string State;
    string ZipAddon;
    string Zip;
    string Addon;
    string RecordType;
    string CarrierRoute;
    string FirmRecepient;
    string Urbanization;
    string FinanceNumber;
    string ErrorDesc;
    int SearchLeft;
    string Score;
}
```

Example

Input:

```xml
<username>username</username>
<password>password</password>
<firmname>Intelligent Search Technology</firmname>
<urbanization> </urbanization>
<delivery_line_1>Mill Town Road</delivery_line_1>
<delivery_line_2> </delivery_line_2>
<city_state_zip>NY 10509</city_state_zip>
```
wsGetAccountAccess

Description
Returns information about account access privileges.

XML template

```xml
<username>string</username>
<password>string</password>
```

Parameters

- **username**: account username
- **password**: account password

Example

Input:

```xml
<username>username</username>
<password>password</password>
```

Output:

Output is an object of type WsAccountAccess:

```csharp
class WsAccountAccess {
  bool Ca;
  bool Geo;
  bool Dpv;
  bool LACSLink;
  bool OfficeAddin;
  bool WebClient;
  string CurrentDate;
}
```
**wsGetAccountInfo**

**Description**

Returns account related information. Currently, it is used to retrieve remaining number of transactions. Return codes are:

- 0 – information retrieved successfully
- 1 – invalid account
- 2 – account is disabled
- 3 – account does not have access to CorrectAddress® XML web services
- 4 – unspecified error

**XML template**

```xml
<username>string</username>
<password>string</password>
```

**Parameters**

- **username**
  account username
- **password**
  account password

**Example**

**Input:**

```xml
<username>username</username>
<password>password</password>
```

**Output:**

Output is an object of type WsAccountInfo:

```java
class WsAccountInfo {
    int SearchesLeft;
    int ReturnCode;
}
```
wsCreateBatch

Description

Creates user batch space. The batch space stores all settings necessary to produce a USPS Postal Form 3553 (CASS report) upon completion of a batch job. The following functions allow users to append current transaction to a specific batch space by passing its name in the "batch name" parameter: wsCorrectA, wsTigerCA. Returns boolean ‘true’ if the batch was created successfully; otherwise boolean ‘false’ is returned.

Note: The batch space only stores job statistics, *not* the actual output address data. Storing address data returned by web services is users’ responsibility.

XML template

```xml
<username>string</username>
<password>string</password>
<batchname>string</batchname>
```

Parameters

- **username**: account username
- **password**: account password
- **batchname**: batch space name
wsDeleteBatch

Description
Empties and removes specified user batch space. Like the CreateBatch function, a boolean ‘true’ is returned if the batch was deleted successfully; otherwise boolean ‘false’ is returned.

XML template
Same as in CreateBatch Function.

Parameters
Same as in CreateBatch Function.
wsGetBatchList

Description

Returns all current batch spaces under user’s account as a string array.

XML template

```
<username>string</username>
<password>string</password>
```

Parameters

- **username**: account username
- **password**: account password
wsGetPS3553form

Description
Retrieves current batch job statistics necessary to generate USPS Postal Form 3553 (CASS report). Data is returned in an XML format.

XML template
Same as in wsCreateBatch Function.

Parameters
Same as in wsCreateBatch Function.

Output:
Output is an object of type WsPS3553:

class WsPS3553{
    string CADate;
    string Company;
    string Software;
    string Configuration;
    int RecordCount;
    int ZipCount;
    int AddonCount;
    int CRouteCount;
    int LOTCount;
    int DPCCount;
    int LACSCount;
    int EWSCount;
    int DPVCount;
    int HDefaultCount;
    int HExactCount;
    int RRDefaultCount;
    int RRExactCount;
    int RDICount;
    string Message;
}

CADate records the date of the USPS data used for processing.

Company, Software, and Configuration record information about organization and the CASS™-certified software. The rest are counters based on input data. RecordCount records the overall number of records processed for this particular batch job.
The following are the fields in PS3553 that must be filled out with information returned by the web service:

Section A (Software):
1. [Company]
2. [Software]
3. [Configuration]
7. [Company]
8. [Software]
9. [Configuration]

Section B (List):
2a. Current date
2c. Current date
3a. [CADate]
3c. [CADate]
5. 1
6. [RecordCount]

Section C (Output):
1a. [AddonCount]
1b. 0
1c. [DPCCount]
1d. [ZipCount]
1e. [CRouteCount]
1f. [LOTCount]
2a. From [CADate] To [CADate + 180 days]
2c. From [CADate] To [CADate + 180 days]
  2d. From [CADate] To [CADate + 365 days]
  2e. From [CADate] To [CADate + 90 days]
2f. From [CADate] To [CADate + 90 days]

Section D (Mailer):

Section E (QSS):
High Rise Default = [HDefaultCount]
High Rise Exact = [HExactCount]
RR Default = [RRDefaultCount]
RR Exact = [RRExactCount]
LACS = [LACSCount]
EWS = [EWSCount]
DPV = [DPVCount]
RDI = [RDICount]

RDICount (Residential Delivery Indicator) is always set to 0. This functionality is currently not available via CorrectAddress® web services.

Message will return the name of the batch job if data retrieval is successful. Otherwise, it will return an error message.

Example
Input:

```xml
<username>username</username>
<password>password</password>
<city>testBatch</city>
```
GetMuniProv

Description
Accepts Canadian postal code as input.
Returns municipality and province information.
Number of remaining account transactions is returned in the SearchLeft output field.
Any errors are returned in the ErrorDesc output field.

XML template

```
<username>string</username>
<password>string</password>
<postalCode>string</postalCode>
```

Parameters

username  
account username

password  
account password

postalCode  
input Canada Post postal code

Output:
Output is an object of type GetMuniProv:

```
class GetMuniProv{
    string PostalCode;
    string Province;
    string Municipality;
    int SearchLeft;
    string ErrorDesc;
}
```

Example
Input:

```
<username>username</username>
<password>password</password>
<postalCode>T0J0A0</postalCode>
```
Chapter 3 - Web Service Integration

C#

1. Generate the C# stubs from the WSDL as follows:
   ```
   wsd1.exe /n:CorrectAddressWSDemo <WSDL/URL>
   ```
   The tool will indicate where the resultant file was created
2. Add a reference to CorrectAddressWebService.cs (which is the file created in the step above)
3. Ensure that the project references ‘System.Web’ and ‘System.Web.Services’
4. Call the CorrectAddress web service

Example of calling `wsCorrectA`:

```csharp
string username = "your username";
string password = "your password";
WsCorrectAddress[] caResultList = caWebService.wsCorrectA(username, password, "Intelligent Search Technology Ltd.", "", "445 Hamilton Ave, ste 608", "", "10601", "", "", "");
```

Sample C# project and supplementary documentation is available at:

https://www.intelligentsearch.com/CorrectAddressWS/docs/CSharp/Example.zip

The example above uses the WSDL utility to generate client stubs. As an alternative, users may add a direct web reference to the web service by using the Add Web Reference tool in the Visual Studio project.
Example of calling **wsCorrectA**: 

```cfscript>
//variable definitions
username = "your username";
password = "your password";
batchname = "test_batch";

wsdl = "https://www.intelligentsearch.com/CorrectAddressWS/CorrectAddressWebService.asmx?wsdl";

// create a webservice object
ws = createObject("webservice", wsdl);

// Sample wsCorrectA call
ws_result = ws.wsCorrectA(username, password, "Intelligent Search Technology Ltd.", ", ", "445 Hamilton Ave, Ste 608", ", ", "10601", ", ", ", ", batchname);

ca_results = ws_result.getWsCorrectAddress();
</cfscript>

Sample ColdFusion script and supplementary documentation is available at: [https://www.intelligentsearch.com/CorrectAddressWS/docs/ColdFusion/Example.zip](https://www.intelligentsearch.com/CorrectAddressWS/docs/ColdFusion/Example.zip)
JAVA

1. Generate the Java stubs from the WSDL as follows:
   
   ```
   wsimport -s src -d bin <url_to_wsdl>
   ```
   
   This will create the source files in the /src directory and class files in the /bin directory.

2. Import the source files generated in step 1 into the project.

3. Call the CorrectAddress web service

Example of calling `wsCorrectA`:

```java
// 1. Define an object of the 'web service' class
CorrectAddressWebService caWebServices;
// 2. Define an object of the 'soap' class
CorrectAddressWebServiceSoap soap;
// 3. Factory object to 'create' other objects
ObjectFactory factory;

// Account Specific info
final String username = "your username";
final String password = "your password";
final String batchName = "test batch";
// 4. Create holder for the CorrectAddress Response
WsCorrectAResponse wsCorrectResult = factory.createWsCorrectAResponse();

// 5. Call the desired web service
wsCorrectResult.setWsCorrectAResult(soap.wsCorrectA(username, password, " ",
" ", "445 Hamilton Ave", "Ste 608", "10601", " ", " ", " ");
```

Sample Java script and supplementary documentation is available at:

https://www.intelligentsearch.com/CorrectAddressWS/docs/Java/

Example.zip
ORACLE

1. Create the PL SQL package as defined in soap_api.pls (located in the example package below) as follows:

   @soap_api.pls

2. Call the CorrectAddress web service

3. Use the SOAP API to parse the output fields, as necessary.

Sample Oracle script and supplementary documentation is available at:
https://www.intelligentsearch.com/CorrectAddressWS/docs/Oracle/Example.zip
PHP

Example of calling **wsCorrectA**: 

```php
$wsdl="https://www.intelligentsearch.com/CorrectAddressWS/CorrectAddressWebService.asmx?WSDL";
$client=new nusoap_client($wsdl, 'wsdl');
$address_params=arRay(
    'username'=> 'your username',
    'password'=> 'your password',
    'firmname'=> 'Intelligent Search Technology Ltd.',
    'urbanization'=> '',
    'delivery_line_1'=> '445 Hamilton Ave, ste 608',
    'delivery_line_2'=> '',
    'city_state_zip'=> '10601',
    'ca_codes'=> '',
    'ca_filler'=> '',
    'batchname'=> 'test_batch'
);
```

Sample PHP script and supplementary documentation is available at:

https://www.intelligentsearch.com/CorrectAddressWS/docs/PHP/

*Note:* The examples use a group of classes called NuSOAP that facilitate creation and consumption of SOAP web services. More information along with the actual PHP classes can be found at https://sourceforge.net/projects/nusoap/ NuSOAP can be replaced by any other SOAP API.
VB.NET

1. Generate the VB.NET stubs from the WSDL as follows:
   
   \[\text{wsdl.exe /l:VB /n:CorrectAddressWSDemo <WSDL/URL>}\]

   The tool will indicate where the resultant file was created

2. Add a reference to CorrectAddressWebService.vb (which is the file created in the step above)

3. Ensure that the project references 'System.Web' and 'System.Web.Services'

4. Call the CorrectAddress web service

Example of calling \texttt{wsCorrectA}:

```vbnet
Dim username As String = "your username"
Dim password As String = "your password"

Dim caResultList As WsCorrectAddress() = caWebService.wsCorrectA(username, password, "Intelligent Search Technology Ltd.", " ", "445 Hamilton Ave, ste 608", " ", "10601", " ", " ", " ")
```

Sample VB.NET project and supplementary documentation is available at:

[https://www.intelligentsearch.com/CorrectAddress_WS/docs/VB.NET/](https://www.intelligentsearch.com/CorrectAddress_WS/docs/VB.NET/)

\[\text{Note}\] The example above uses the WSDL utility to generate client stubs. As an alternative, users may add a direct web reference to the web service by using the Add Web Reference tool in the Visual Studio project.
Example of calling **wsCorrectA**: 

```vbscript
<%  
  Dim requestDoc  
  Dim webXML  
  Dim XMLresponse  
  Dim objXMLHTTP  
  Dim CAResult  
  set requestDoc=Server.CreateObject("Microsoft.XMLDOM")  
  requestDoc.async = false  
  set objXMLHTTP = Server.CreateObject("Microsoft.XMLHTTP")  
  webXML = "<username >USERNAME</username>" & "<password>PASSWORD</password>" & "<firmname>Intelligent Search Technology Ltd.</firmname>" & "<urbanization>Urbanization</urbanization>" & "<delivery_line_1>445 Hamilton Ave, Ste 608</delivery_line_1>" & "<delivery_line_2>" & "<city_state_zip>White Plains, NY 10601</city_state_zip>" & "<ca_codes>McRy </ca_codes>" & "<ca_filler>" & "</ca_filler>" & "<batchname></batchname>"  
  CAResult = ""  
  CAResult = CAResult & "<?xml version="1.0" encoding="utf-8"?>" & vbCrLf  
  CAResult = CAResult & "<soap:Body>" & vbCrLf  
  CAResult = CAResult & "<wsCorrectA xmlns="https://www.intelligentsearch.com/HostedWebServices/">" & vbCrLf  
  CAResult = CAResult & webXML  
  CAResult = CAResult & "</wsCorrectA>" & vbCrLf  
  CAResult = CAResult & "</soap:Body>" & vbCrLf  
  CAResult = CAResult & "</soap:Envelope>" & vbCrLf  
  objXMLHTTP.open "post", "https://www.intelligentsearch.com/CorrectAddressWS/CorrectAddressWebService.asmx", false  
  objXMLHTTP.setRequestHeader "Content-Type", "text/xml; charset=utf-8"  
  objXMLHTTP.setRequestHeader "Content-Length", Len(CAResult)  
  objXMLHTTP.setRequestHeader "SOAPAction", "https://www.intelligentsearch.com/HostedWebServices/wsCorrectA"  
  objXMLHTTP.send CAResult  
  requestDoc.loadXML objXMLHTTP.responseText  
  XMLresponse = requestDoc.text  
  response.write XMLresponse  
%>
```

Sample Classic ASP script and supplementary documentation is available at: [https://www.intelligentsearch.com/CorrectAddress_WS/docs/ClassicASP/Example.zip](https://www.intelligentsearch.com/CorrectAddress_WS/docs/ClassicASP/Example.zip)
Appendix A - Transitioning to the New Object Model

This appendix provides guidance for users transitioning to the new web service model and those using one of the functions deprecated on or before January 1, 2010.

The latest version of the API incorporates a number of enhancements. To ensure smooth transition to the object-based model, several functions were renamed and a few were deprecated.

Below is the list of web service name changes in effect January 1, 2010 (two primary validation service names – wsCorrectA and wsTigerCA remain the same):

<table>
<thead>
<tr>
<th>FORMER NAME</th>
<th>NEW NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>CallFindCityCounty</td>
<td>wsFindCityCounty</td>
</tr>
<tr>
<td>CallFindZipCity</td>
<td>wsFindZipCity</td>
</tr>
<tr>
<td>CallFindCityState</td>
<td>wsFindCityState</td>
</tr>
<tr>
<td>CallStreetSearch2</td>
<td>wsStreetSearch2</td>
</tr>
<tr>
<td>getAccountInfo</td>
<td>wsGetAccountInfo</td>
</tr>
<tr>
<td>CreateBatch</td>
<td>wsCreateBatch</td>
</tr>
<tr>
<td>DeleteBatch</td>
<td>wsDeleteBatch</td>
</tr>
<tr>
<td>getBatchList</td>
<td>wsGetBatchList</td>
</tr>
<tr>
<td>getPS3553form</td>
<td>wsGetPS3553form</td>
</tr>
</tbody>
</table>

If the function you are currently using is no longer listed and has been deprecated, please refer to the table below for the new function name to use (see Chapter 2: Functions Specifications earlier in the document):

<table>
<thead>
<tr>
<th>DEPRECATED FUNCTION NAME</th>
<th>NEW FUNCTION TO USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>callCA</td>
<td>wsCorrectA</td>
</tr>
<tr>
<td>callCA2</td>
<td>wsCorrectA</td>
</tr>
<tr>
<td>callCAstd</td>
<td>wsCorrectA</td>
</tr>
<tr>
<td>correctA</td>
<td>wsCorrectA</td>
</tr>
<tr>
<td>correctA2</td>
<td>wsCorrectA</td>
</tr>
<tr>
<td>CallStreetSearch</td>
<td>wsStreetSearch2</td>
</tr>
<tr>
<td>TigerCA</td>
<td>wsTigerCA</td>
</tr>
<tr>
<td>getPS3553form2</td>
<td>wsGetPS3553form</td>
</tr>
</tbody>
</table>