Reference Guide

WebSmart Address Verifier

MELISSA DATA®

WebSmart Address Verifier

Reference Guide

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
Web site: www.MelissaData.com
For the latest version of this Reference Guide, visit
http://www.MelissaData.com/tech/websmart.htm.

Document Code: WSARFG
Revision Number: 131021.032
Last Update: October 21, 2013

Dear Programmer,

I would like to take this opportunity to introduce you to Melissa Data Corp. Founded in 1985, Melissa Data provides data quality solutions, with emphasis on address and phone verification, postal encoding, and data enhancements.

We are a leading provider of cost-effective solutions for achieving the highest level of data quality for lifetime value. A powerful line of software, databases, components, and services afford our customers the flexibility to cleanse and update contact information using almost any language, platform, and media for point-of-entry or batch processing.

This online manual will guide you through the properties and methods 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 ray@MelissaData.com.

I look forward to hearing from you.

Best Wishes,

Raymond F. Melissa
President

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 Address Verifier

The WebSmart Address Verifier service helps to guarantee that your address data contains a properly coded and standardized address.

The Address Verifier:

- Verifies that the record contains a deliverable address.
- Standardizes the street address using the preferred USPS abbreviations.
- Flags undeliverable or incomplete addresses.
- Applies the correct ZIP + 4 if missing.
- Appends Carrier Route and Delivery Point numbers to the address record.
- Optionally parses the street address.

Address Handling

A key concept for Address Verifier is understanding how the service handles address data. Address Verifier accepts two lines of street address information, the AddressLine1 and AddressLine2 elements in each record of the Request Array.

AddressLine1 will typically contain the primary street address (Street number, street name, plus any directionals and street suffixes). It may or may not contain also a secondary address, such as a suite number, address number, unit number, or a private mailbox located at a commercial mail receiving agency (CMRA).

AddressLine2 will often contain the secondary address information, if it is not submitted as part of AddressLine1 or included in the Suite or PrivateMailBox elements. This element may also be used if there is another primary address, either a Post Office Box or a separate street address, that is part of the same record.

If the secondary Address information is submitted in the AddressLine2 element, Address Verifier will append this information to AddressLine1 before processing the record.

Example #1

If the following were submitted with the request:

AddressLine1: 1234 Main Street
AddressLine2: Suite #101

This is the address that would actually be verified:

AddressLine1: 1234 Main Street Suite #101
AddressLine2: <empty>

If there is an additional primary address in AddressLine2, such as a P.O. Box or second street address, and the contents of AddressLine1 cannot be verified, then Address Verifier will attempt to verify the second line.

Example #2

If the following were submitted with the request:

AddressLine1: 1234 Main Street
AddressLine2: P.O. Box 101

Assuming that AddressLine1 didn't contain a verifiable address, this is what Address Verifier would consider the contents of AddressLine2.

If the AddressLine2 contains a verifiable address, then the contents of that element will be used. In that case, Address Verifier will swap the contents of AddressLine1 and AddressLine2 and the response array will return the contents of these elements in this order:

Address1: P.O.Box 101
Address2: 1234 Main Street

If the contents of neither element can be successfully verified, then Address Verifier will flag the error and the contents of AddressLine1 and AddressLine2 will be returned in their **original** order:

Address1: 1234 Main Street
Address2: P.O.Box 101

Secondary Addresses

Secondary addresses include suite numbers, unit numbers and residential apartment numbers. It could also referred a private mailbox (PMB) at a Commercial Mail Receiving Agency (CMRA).

The secondary address can be passed to Address Verifier at the end the first address line, as the second address line or via the suite element.

The National Postal database identifies certain primary addresses as highrises, business parks and apartment buildings. Therefore, Address Verifier would be able to assign the correct secondary address designator to the following address:

1234 Main St #101

For example, if the primary address were an apartment complex, Address Verifier would return “Apt 101” as the suite information.

CMRAs like the UPS Store and other mailbox stores are a special case. These are often located at shopping centers and the store itself will have a suite number. This means that addresses located at a CMRA will often have both a suite number and a PMB number, like this:

1234 Main Street
Suite C1 PMB#101

CMRAs are identified in the national address database as this kind of business.

Example:

1234 Main Street #101

Assuming that 1234 Main Street is identified in the national database as a CMRA, the “#101” portion would be returned by the Private Mail Box element rather than the Suite element.

Be aware that Address Verifier will always treat a second item of secondary address information as a PMB number.

Example:

1234 Main Street
Suite C1 #101

Whether or not the primary address is identified in the database as a CMRA, Address Verifier will identify the “#101” portion of the second line as a PMB number and return this number in the Private Mailbox element.

Even if an address is identified as a CMRA, if a secondary address is explicitly identified as a suite or anything other than private mailbox, this information will be treated as a suite and not a PMB.

Example:

1234 Main Street
Suite 101

Assuming that the primary address belongs to a CMRA, because the secondary address was supplied as “Suite 101,” this will be treated as a suite number and not a PMB number. If the number 101 was meant to refer to a PMB and there is not Suite 101 at this address, this would probably cause the address to fail verification.

City or Postal Code Information

In order to verify any address, Address Verifier requires information on the city, the state or province, and a ZIP or Postal code. With the city and the state/province, the Address Verifier will determine the correct ZIP/Postal code and use this to verify the address.

Conversely, if supplied with a correct ZIP/Postal code, Address Verifier can look up the city and the state/province.

Therefore either the ZIP/Postal code or the city plus the state/province are required. If the supplied city and state/province do not match the ZIP/Postal code, Address Verifier will use the city and state/province to look up the ZIP/Postal code.

Adding Address Verifier 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://addresscheck.melissadata.net/v2/SOAP/Service.svc
```

If you are not using Visual Studio .NET, see the documentation 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://addresscheck.melissadata.net/v2/XML/Service.svc/
   doAddressCheck
```

Building a REST Request

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

```
https://addresscheck.melissadata.net/v2/REST/Service.svc/
   doAddressCheck
```

3 Address Verifier Request

At the very minimum, a request to the WebSmart Address Verifier consists of the user's Customer ID and at least one address record.

SOAP Request

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

Step 1 – Create the Request and Response Objects

```
Dim ReqAddressCheck As New dqwsAddressCheck.RequestArray
Dim ResAddressCheck As New dqwsAddressCheck.ResponseArray
```

Step 2 – Assign the General Request Values

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

```
ReqAddressCheck.CustomerID = strCustID
ReqAddressCheck.TransmissionReference = strTranRef
ReqAddressCheck.OptAddressParsed = True
```

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

Step 3 – Dimension the Record Array

The maximum number of records per request is 100, therefore the largest dimension will be 99.

```
ReDim ReqAddressCheck.Record(99)
```

For maximum efficiency, you should dimension the array using the exact number of records being submitted minus one.

Step 4 – Build the Record Array

The exact method for building the array will depend on the exact database software in use, but you will need to loop through every record to be submitted and assign the required values to the corresponding elements for each record in the RequestArray.

```
ReqAddressCheck.Record(intRecord) = New
   dqwsAddressCheck.RequestArrayRecord
ReqAddressCheck.Record(intRecord).AddressLine1 = "22382
   Avenida Empresa"
ReqAddressCheck.Record(intRecord).Zip = "92688"
ReqAddressCheck.Record(intRecord).RecordID = 1
```

The lines above show only the elements that are absolutely required to submit a record to the web service. See the rest of this chapter for a description of all of the elements available to include with a request record.

Repeat for each record being submitted with the current RequestArray.

Step 5 – Submit the Request Array

The final step is to create the Service Client Object and then submit the RequestArray object doAddressCheck method. This sends the data to the web service and retrieves the ResponseArray object.

```
AddressCheckClient = New dqwsAddressCheck.Service
ResAddressCheck =
   AddressCheckClient.doAddressCheck(ReqAddressCheck)
AddressCheckClient.Dispose()
```

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://addresscheck.melissadata.net/v2/XML/Service.svc/
   doAddressCheck
```

Rather than an array of Record object, an XML request contains a `<Record>` element for each address record, up to 100.

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

```
<RequestArray>
   <TransmissionReference>Web Service Test 2008/12/31
   </TransmissionReference>
   <CustomerID>123456789</CustomerID>
   <OptAddressParsed>True</OptAddressParsed>
   <Record>
      <RecordID>1</RecordID>>
      <Company />
      <LastName />
      <Urbanization />
      <AddressLine1>22382 Avenida Empresa</AddressLine1>
      <AddressLine2 />
      <Suite />
      <City>Rancho Santa Margarita</City>
      <State>CA</State>
      <Zip>92688</Zip>
      <Plus4 />
      <Country />
   </Record>
   <Record>
   ...
   </Record>
</RequestArray>
```

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://addresscheck.melissadata.net/v2/REST/Service.svc/
   doAddressCheck?id=12345678&opt=true&a1=22382%20Avenida
   %20Empresa&city=Rancho%20Santa%20Margarita&state=CA&zi
   p=92688
```

The record ID element does not exist for the REST interface, since you can only submit a single record per request.

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

```
<RequestArray>
   <CustomerID>String</CustomerID>
</RequestArray>
```

REST

```
id={CustomerID}
```

optAddressedParsed

This is an optional string value that controls whether or not the Response Array returns the parsed address data.

Remarks

If you send a string value of “True” with this element, the Address Verifier service will parse the street address into its component parts.

If this element is not set to “True,” none of the child elements of <Parsed> will be populated for any record in the Response Array.

Syntax

SOAP

```
Request.OptAddressParsed = string
```

XML

```
<RequestArray>
   <OptAddressParsed>String</OptAddressParsed>
</RequestArray>
```

REST

```
opt={OptAddressParsed}
```

Transmission Reference

This is an optional string value that may be passed with each Request Array 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

```
<RequestArray>
    <TransmissionReference>String</TransmissionReference>
</RequestArray>
```

REST

```
t={transMissionReference}
```

Record Elements

For the SOAP and XML web services, the Request Array will contain an element or property called Record. In SOAP this property is an array of object variables of the type Record. XML will have as many Record elements as there are addresses being submitted to the web service.

The REST interface only allows a single record per request.

Record ID

This element is a string value containing a unique identifier for the current record.

Remarks

Use this element to match the record with the record returned with the Response Array.

When using the SOAP interface, if this element is not populated, the web service will automatically insert a sequential number for each record.

There is no equivalent for Record ID for the REST interface.

Syntax

SOAP

```
Request.Record().RecordID = string
```

XML

```
<RequestArray>
    <Record>
        <RecordID>String</RecordID>
    </Record>
</RequestArray>
```

Company

This is an optional string value containing any company name associated with the current address record.

Remarks

If a company name is supplied for a company that has been assigned a specific Plus4 by the USPS, the address checking logic will return a more accurate Plus4 code.

If a company name is not supplied, the address checking logic will still be able to code the address but it will supply a more generic “street level default” Plus4.

The Company element is also used by the SuiteLink functionality of the Web Service. SuiteLink will append missing suite information to addresses that are associated with the Company Name of the input address. Addresses that have their Suite Information populated by SuiteLink will have a Result code of AS14.

Syntax

SOAP

```
Request.Record().Company = string
```

XML

```
<RequestArray>
    <Record>
        <Company>String</Company>
    </Record>
</RequestArray>
```

REST

```
comp={Company}
```

Urbanization

This is an optional string value containing the Urbanization name for the current address record. This element applies only to Puerto Rican addresses.

Remarks

The Urbanization Property is an optional element set by the user. It is only used when attempting to correct addresses in Puerto Rico. If it is not set, the address checking logic will still be able to code some records, but it may produce more multiple matches than usual for Puerto Rican addresses. This happens because the urbanization name is used to break ties when a ZIP Code is linked to multiple instances of the same address.

The urbanization name tells the address checking logic which “neighborhood” to look in if more than one likely address candidate is found.

If just one address is found, the address checking logic can correct the address and return the urbanization name.

Syntax

SOAP

```
Request.Record().Urbanization = string
```

XML

```
<RequestArray>
   <Record>
      <Urbanization>String</Urbanization>
   </Record>
</RequestArray>
```

REST

```
u={Urbanization}
```

AddressLine1

This is a required string value containing the first line of the street address from the current address record.

Remarks

This element must be passed with each record and contain the primary street address for that record. It may also contain the suite or other secondary address information, if this is not sent separately with those respective elements.

Syntax

SOAP

```
Request.Record().AddressLine1 = string
```

XML

```
<RequestArray>
   <Record>
      <AddressLine1>String</AddressLine1>
   </Record>
</RequestArray>
```

REST

```
a1={AddressLine1}
```

AddressLine2

This is an optional string value containing the second line of the street address, if any, from the current address record.

Remarks

This element is optional. It can contain either a suite, apartment or unit number. It may also a different primary address such as a P.O. Box.

To see how Address Verifier handles information found in AddressLine2, see the section called *Address Handling* on page 5 of this manual.

Syntax

SOAP

```
Request.Record().AddressLine2 = string
```

XML

```
<RequestArray>
   <Record>
      <AddressLine2>String</AddressLine2>
   </Record>
</RequestArray>
```

REST

```
a2={AddressLine2}
```

Suite

This is an optional string value containing the suite name and number for the current record.

Remarks

Use this element to send the suite number if that information is stored separately from the street address in your original data.

The suite information will also be detected if it is part of the AddressLine1 element or stored in AddressLine2. Use this element if your original database table has an explicit Suite element of its own.

Syntax

SOAP

```
Request.Record().Suite = string
```

XML

```
<RequestArray>
    <Record>
        <Suite>String</Suite>
    </Record>
</RequestArray>
```

REST

```
ste={suite}
```

City

This is a required string value containing the city or municipality name for the current record.

Remarks

When submitting an address record, you have the option of using either City and State or at least five-digit ZIP Code or six-character Postal Code. For example, if you set City as “Rancho Santa Margarita” and State as “CA,” that would be the same as setting the Zip element to “92688.”

You must at least set both the City and State element OR the Zip element.

If you set all three elements and the contents of the Zip element are not correct for values for the City and State elements, the web service will use the City and State elements to determine the correct ZIP Code.

Syntax

SOAP

```
Request.Record().City = string
```

XML

```
<RequestArray>
    <Record>
        <City>String</City>
    </Record>
</RequestArray>
```

REST

```
city={City}
```

State

This is a required string value containing the name or abbreviation for the state or province for the current record.

Remarks

When submitting an address record, you have the option of using either City and State or at least the five-digit ZIP Code or six-character Postal Code. For example, if you set City as “Rancho Santa Margarita” and State as “CA,” that would be the same as setting the Zip element to “92688.”

You must at least set both the City and State element OR the Zip element.

If you set all three elements and the contents of the Zip element are not correct for values for the City and State elements, the web service will use the City and State elements to determine the correct ZIP Code.

Syntax

SOAP

```
Request.Record().State = string
```

XML

```
<RequestArray>
    <Record>
        <State>String</State>
    </Record>
</RequestArray>
```

REST

```
state={State}
```

Zip

This is a required string value containing ZIP or Postal code for the current record.

Remarks

When submitting an address record, you have the option of using either City and State or at least the five-digit ZIP Code or six-character Postal Code. For example, if you set city as “Rancho Santa Margarita” and State as “CA,” that would be the same as setting the Zip element to “92688.”

You must at least set both the City and State element OR the Zip element.

If you set all three elements and the ZIP Code is not correct for the City and State, the web service will use the City and State elements to determine the correct ZIP Code.

Syntax

SOAP

```
Request.Record().Zip = string
```

XML

```
<RequestArray>
    <Record>
        <Zip>String</Zip>
    </Record>
</RequestArray>
```

REST

```
zip={Zip}
```

Country

This is an optional string value containing the name or abbreviation of the country for the current address record.

Remarks

The web service can only verify addresses in the United States or Canada.

Syntax

SOAP

```
Request.Record().Country = string
```

XML

```
<RequestArray>
   <Record>
      <Country>String</Country>
   </Record>
</RequestArray>
```

REST

```
ctry={Country}
```

LastName

Sets the last name associated with a residential address. This property is required to use the AddressPlus functionality.

Remarks

AddressPlus enables Address Check to assign secondary numbers (Apartment numbers, unit numbers for certain residential addresses. In addition to the basic address information (the Address element plus the City, State and Zip element), you must also submit the correct last name associated with the address record to enable the service to distinguish the record from other suites at the same address. If an addresses have this Information appended by AddressPlus, the Response array will return a result code of “AS15.”

AddressPlus is only available for U.S. addresses.

Syntax

SOAP

```
Request.Record().LastName = string
```

XML

```
<RequestArray>
    <Record>
        <LastName>String</LastName>
    </Record>
</RequestArray>
```

REST

```
last={LastName}
```

4 Address Verifier Response

The SOAP interface for the Address Verifier service returns a ResponseArray Object. The primary component of this object is an array of Record objects, one for each record submitted with the RequestArray, containing the verified and standardized address data.

The XML interface returns an XML document containing a number of <Record> elements, one for each record submitted with the Request, containing the verified and standardized address data.

The REST interface returns an XML document with a single <Record> element.

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 records returned with the current response.

Remarks

This property returns the number of 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 and system error messages from the most recent request sent to the service.

Remarks

Do not confuse this element with the Results element returned with each record, described on page 34. This element returns error messages caused by the most recent request as a whole.

The possible values are:

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.

Syntax

SOAP

```
string = Response.Results
```

XML

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

Version

Returns a string value containing the current version number of the Address Verifier service.

Syntax

SOAP

```
string = Response.Version
```

XML

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

Record Elements

The SOAP version of the Response Array returns a property called Record which is an array of Record objects, one for each record submitted with the original Request Array.

The XML service returns one <Record> element for every record submitted with the original request.

The REST response is identical to the XML response, but will only contain a single <Record> element.

The following section describes the elements returned by each record in the Response Array.

Record ID

For each record in the Response Array, this element returns a string value containing the unique identifier for the current record if one was passed to the Request Array.

Remarks

Use this element to match the record in the Response Array with the record originally passed with the request.

Syntax

SOAP

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

XML

```
<ResponseArray>
    <Record>
        <RecordID>String</RecordID>
    </Record>
</ResponseArray>
```

Results

For each record in the Response Array, returns a string value containing status and error codes for the current record. Multiple codes are separated by commas.

Remarks

This element returns the status and error messages for each record in the Response Array. For the general status and error messages generated by the most recent AddressCheck request, see the general Results element on page 30.

The Result element may return one or more four-character strings, separated by commas, depending on the result generated by the current record.

If the address in the current record was verified, this element will contain the value “AS01” at the very minimum and may include more of the “AS” codes. If the address could not be verified, the codes beginning with “AE” will indicate the reason or reasons why verification failed.

The possible values are:

Code	Short Description	Long Description
		Status Result Codes
AS01	Address Fully Verified	The address is valid and deliverable according to official postal agencies.
AS02	Street Only Match	The street address was verified but the suite number is missing or invalid.
AS03	Non USPS Address Match	US Only. This US address is not serviced by the USPS but does exist and may receive mail through third party carriers like UPS.
AS09	Foreign Address	The address is in a non-supported country.
AS10	CMRA Address	US Only. The address is a Commercial Mail Receiving Agency (CMRA) like a Mailboxes Ect. These addresses include a Private Mail Box (PMB or #) number.
AS13	Address Updated By LACS	US Only. The address has been converted by LACSLink® from a rural-style address to a city-style address.

Code	Short Description	Long Description
AS14	Suite Appended	US Only. A suite was appended by SuiteLink™ using the address and company name.
AS15	Apartment Appended	An apartment number was appended by AddressPlus using the address and last name.
AS16	Vacant Address	US Only. The address has been unoccupied for more than 90 days.
AS17	No Mail Delivery	US Only. The address does not currently receive mail but will likely in the near future.
AS18	DPV Locked Out	US Only. DPV processing was terminated due to the detection of what is determined to be an artificially created address.
AS20	Deliverable only by USPS	US Only. This address can only receive mail delivered through the USPS (ie. PO Box or a military address).
AS23	Extraneous Information	Extraneous information not used in verifying the address was found. This has been placed in the ParsedGarbage field.
		Error Result Codes
AE01	Postal Code Error	The Postal Code does not exist and could not be determined by the city/municipality and state/province.
AE02	Unknown Street	Could not match the input street to a unique street name. Either no matches or too many matches found.
AE03	Component Mismatch Error	The combination of directionals (N, E, SW, etc) and the suffix (AVE, ST, BLVD) is not correct and produced multiple possible matches.
AE04	Non-Deliverable Address	US Only. A physical plot exists but is not a deliverable addresses. One example might be a railroad track or river running alongside this street, as they would prevent construction of homes in that location.
AE05	Multiple Match	The address was matched to multiple records. There is not enough information available in the address to break the tie between multiple records.

Code	Short Description	Long Description
AE06	Early Warning System	US Only. This address currently cannot be verified but was identified by the Early Warning System (EWS) as containing new streets that might be confused with other existing streets.
AE07	Missing Minimum Address	Minimum requirements for the address to be verified is not met. Address must have at least one address line and also the postal code or the locality/ administrative area.
AE08	Sub Premise Number Invalid	The thoroughfare (street address) was found but the sub premise (suite) was not valid.
AE09	Sub Premise Number Missing	The thoroughfare (street address) was found but the sub premise (suite) was missing.
AE10	Premise Number Invalid	The premise (house or building) number for the address is not valid.
AE11	Premise Number Missing	The premise (house or building) number for the address is missing.
AE12	Box Number Invalid	The PO (Post Office Box), RR (Rural Route), or HC (Highway Contract) Box numer is invalid.
AE13	Box Number Missing	The PO (Post Office Box), RR (Rural Route), or HC (Highway Contract) Box number is missing.
AE14	PMB Number Missing	US Only. The address is a Commercial Mail Receiving Agency (CMRA) and the Private Mail Box (PMB or #) number is missing.
AE17	Sub Premise Not Required	A sub premise (suite) number was entered but the address does not have secondaries.
		Change Codes
AC01	Postal Code Change	The postal code was changed or added.
AC02	Administrative Area Change	The administrative area (state, province) was added or changed.
AC03	Locality Change	The locality (city, municipality) name was added or changed.
AC04	Alternate to Base Change	US Only. The address was found to be an alternate record and changed to the base (preferred) version.

Code	Short Description	Long Description
AC05	Alias Name Change	US Only. An alias is a common abbreviation for a long street name, such as “MLK Blvd” for “Martin Luther King Blvd.” This change code indicates that the full street name (preferred) has been substituted for the alias.
AC06	Address1/ Address2 Swap	Address1 was swapped with Address2 because Address1 could not be verified and Address2 could be verified.
AC07	Address1 & Company Swapped	Address1 was swapped with Company because only Company had a valid address.
AC08	Plus4 Change	US Only. A non-empty plus4 was changed.
AC09	Dependent Locality Change	US Only. The dependent locality (urbanization) was changed.
AC10	Thoroughfare Name Change	The thoroughfare (street) name was changed due to a spelling correction.
AC11	Thoroughfare Suffix Change	The thoroughfare (street) suffix was added or changed, such as from "St" to "Rd."
AC12	Thouroughfare Directional Change	The thoroughfare (street) pre-directional or post-directional was added or changed, such as from "N" to "NW."
AC13	Sub Premise Type Change	The sub premise (suite) type was added or changed, such as from “STE” to “APT.”
AC14	Sub Premise Number Change	The sub premise (suite) unit number was added or changed.

Using Result Codes in your Program:

These result codes are designed to let you easily determine if an address is “good” or “bad.” A good address will contain AS01, a bad one will not.

If you have other conditions for a good address, you can add them as well. For example, if you are not using USPS and cannot deliver to a PO Box or a Military address, you can exclude records that return the result code “AS12.”

After determining if the address is “good” or “bad,” you can check for errors (AE##) to see what was wrong with the address.

Syntax

SOAP

```
string = Response.Record().Result
```

XML

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

Company

For each record in the Response Array, this element returns the string value containing the contents of the Company element from the matching record from the Request Array.

Remarks

The web service does not populate this element, nor does it modify the contents of the Company element from the Request Array.

Syntax

SOAP

```
string = Response.Record().Result
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Company>String</Company>
        </Address>
    </Record>
</ResponseArray>
```

Urbanization

For each record in the Response 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

Urbanization will only be returned for addresses in Puerto Rico. See page 18 for more information on how Urbanization is used.

Syntax

SOAP

```
string = Response.Record().Address.Urbanization.Name
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <Urbanization>
            <Name>String</Name>
         </Urbanization>
      </Address>
   </Record>
</ResponseArray>
```

Address1

For each record in the Response Array, Address1 returns a string value containing the street address that the web service attempted to verify.

Remarks

This element returns the standardized contents of the AddressLine1 element from the matching record in the Request Array.

If the AddressLine1 element contained any suite or secondary address information, this will be moved to the Suite or PrivateMailBox element. See *Address Handling* on page 5 for more information.

Syntax

SOAP

```
string = Response.Record().Address.Address1
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Address1>String</Address1>
        </Address>
    </Record>
</ResponseArray>
```

Address2

For each record in the Response Array, Address2 returns a string value containing the secondary street address.

Remarks

This element returns the contents of the AddressLine2 element from the matching record in the Request Array.

If the AddressLine2 element contained any suite or secondary address information, this will be moved to the Suite or PrivateMailBox element.

For more information, see *Address Handling* on page 5.

Syntax

SOAP

```
string = Response.Record().Address.Address2
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Address2>String</Address2>
        </Address>
    </Record>
</ResponseArray>
```

Suite

For each record in the Response Array, Suite returns a string value containing the standardized suite information for the current address.

Remarks

If the suite information was included as part of AddressLine1 or AddressLine2 in the Request Array, it will be moved to this property when the record is returned by the Response Array.

When the suite number cannot be verified, it will be passed along unmodified. If the address can be verified down to the suite level, the suite number will also be standardized.

Syntax

SOAP

```
string = Response.Record().Address.Suite
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Suite>String</Suite>
        </Address>
    </Record>
</ResponseArray>
```

Private Mail Box

For each record in the Response Array, PrivateMailBox returns a string value containing a private mailbox number (PMB) for an address assigned to a Commercial Mail Receiving Agency (CMRA).

Remarks

CMRAs are private businesses that provide a mailing address and “post office” box for their customers.

Mail is delivered by the Postal Service to the CMRA, which then distributes the mail to the customer’s private mail box.

For more information on how Address Verifier handles PMB numbers see the section entitled *Secondary Addresses* on page 6.

Syntax

SOAP

```
string = Response.Record().Address.PrivateMailBox
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <PrivateMailBox>String</PrivateMailBox>
      </Address>
   </Record>
</ResponseArray>
```

City

For each record in the Response Array, City Name returns a string value containing the full name of the city or municipality. City Abbreviation returns the official thirteen character shorthand for the city name.

Remark

If the returned City element is longer than 13 letters, the City Abbreviation will return the official abbreviation the post office has associated with that city or municipality name. For example, “Fort Lauderdale,” will return the “Ft Lauderdale” as the City Abbreviation.

If the City Name is 13 letters or shorter, City Abbreviation will contain the full city or municipality name.

Syntax

SOAP

```
string = Response.Record().Address.City.Name
string = Response.Record().Address.City.Abbreviation
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <City>
            <Name>String</Name>
            <Abbreviation>String</Abbreviation>
         </City>
      </Address>
   </Record>
</ResponseArray>
```

State

For each record in the Response Array, State Name returns a string value containing the full name of the state or province. State Abbreviation returns a string value containing the standard two-letter Postal abbreviation for the state or province.

Remarks

If the state name “California,” then Abbreviation would be “CA.” For the Province of “Quebec,” Abbreviation would return “QC.”

If the record in the Request Array did not contain a city or state/province, but only a ZIP or Postal code, the web service uses the ZIP or Postal code to determine the city/municipality and state/province information.

Syntax

SOAP

```
string = Response.Record().Address.State.Name
string = Response.Record().Address.State.Abbreviation
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <State>
                <Name>String</Name>
                <Abbreviation>String</Abbreviation>
            </State>
        </Address>
    </Record>
</ResponseArray>
```

Zip

For each record in the Response Array, Zip returns a string value containing the five-digit ZIP Code or six-character postal code for the address.

Remarks

If the ZIP Code supplied with the Request Array is incorrect or not supplied, the address checking logic will return the correct five-digit ZIP Code or six-character Postal Code for this property as long as the correct city/municipality and state/province were supplied with Request Array record.

Syntax

SOAP

```
string = Response.Record().Address.Zip
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <Zip>String</Zip>
      </Address>
   </Record>
</ResponseArray>
```

Plus4

For each record containing a U.S. address in the Response Array, the Plus4 element returns a string value containing the four-digit portion of the ZIP + 4 for the address.

Remarks

If the address sent with the Request Array is not verified, the web service does not return a Plus4 element for that record, even if one was supplied with the Request Array record.

Syntax

SOAP

```
string = Response.Record().Address.Plus4
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <Plus4>String</Plus4>
      </Address>
   </Record>
</ResponseArray>
```

Carrier Route

U.S. Only

For each record containing a U.S. address in the Response 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.Record().Address.CarrierRoute
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <CarrierRoute>String</CarrierRoute>
      </Address>
   </Record>
</ResponseArray>
```

Delivery Point Code

U.S. Only

For each record containing a U.S. address in the Response Array, DeliveryPointCode returns a string value containing the tenth and eleventh digits of the POSTNet barcode number.

Remarks

The complete twelve-digit POSTNet barcode number consists of the ZIP + 4, minus any dash, the two-digit DeliveryPointCode and the single-digit **DeliveryPointCheckDigit**, described on the next page.

Syntax

SOAP

```
string = Response.Record().Address.DeliveryPointCode
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <DeliveryPointCode>String</DeliveryPointCode>
        </Address>
    </Record>
</ResponseArray>
```

Delivery Point Check Digit

U.S. Only

For each record containing a U.S. address in the Response Array, DeliveryPointCheckDigit returns a string value containing the twelfth digit of the POSTNet barcode number.

Remarks

For more information using DeliveryPointCheckDigit to create a POSTNet barcode, see *Delivery Point Code* on page 50.

Syntax

SOAP

```
string = Response.Record().Address.DeliveryPointCheckDigit
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <DeliveryPointCheckDigit>
                String
            </DeliveryPointCheckDigit>
        </Address>
    </Record>
</ResponseArray>
```

Address Type

U.S. and Canada

For each record containing a U.S. or Canadian address in the Response Array, Address Type Code returns a string value containing the one-character code for the specific type of address returned by the current record. Address Type Description returns a string value containing a description of the address type that corresponds to the Address Type Code.

Remarks

The GetAddressTypeCode returns a one-character string value for the Type and Address that was coded. For U.S. addresses, Address Verifier would return one of the following codes:

Code	Descriptions
A	Alias
F	Firm or Company address
G	General Delivery address
H	Highrise or Business Complex
P	PO Box address
R	Rural Route address
S	Street or Residential address

For Canadian addresses, Address Verifier would return one of the following codes.

Code	Descriptions
1	Street
2	Street Served By Route and GD
3	Lock Box
4	Route Service
5	General Delivery
B	LVR Street
C	Government Street
D	LVR Lock Box
E	Government Lock Box
L	LVR General Delivery
K	Building

Syntax

SOAP

```
string = Response.Record().Address.Type.Address.Code
string = Response.Record().Address.Type.Address.Description
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <Type>
            <Address>
               <Code>String</Code>
               <Description>String</Description>
            </Address>
         </Type>
      </Address>
   </Record>
</ResponseArray>
```

Zip Type

U.S. Only

For each record containing a U.S. address in the Response Array, Zip Type Code returns a string value containing the one-character code for the specific type of ZIP Code for the address returned by the current record. Zip Type Description returns a string value containing a description of the ZIP Code type that corresponds to the Zip Type Code.

Remarks

The Zip Type Code and Zip Type Description elements return one of the following sets of values:

Code	Descriptions
P	PO Box ZIP Code
U	Unique ZIP Code
M	Military ZIP Code
Empty	Standard ZIP Code

The empty string for a Standard ZIP Code is a zero-length string, not a space.

Syntax

SOAP

```
string = Response.Record().Address.Type.Zip.Code
string = Response.Record().Address.Type.Zip.Description
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <Type>
            <Zip>
               <Code>String</Code>
               <Description>String</Description>
            </Zip>
         </Type>
      </Address>
   </Record>
</ResponseArray>
```

Country

For each record in the Response Array, the web service returns two string values designating the country in which the address is located. Country Name returns the full name of the country while Abbreviation returns a two-character abbreviation.

Remarks

At present, the web service only processes U.S. and Canadian addresses, so the possible values for these two elements:

Code	Name
US	United States of America
CA	Canada

Syntax

SOAP

```
string = Response.Record().Address.Country.Abbreviation
string = Response.Record().Address.Country.Name
```

XML

```
<ResponseArray>
   <Record>
      <Address>
         <Country>
            <Abbreviation>String</Abbreviation>
            <Name>String</Name>
         </Country>
      </Address>
   </Record>
</ResponseArray>
```

Address Key

For each record in the Response Array, returns a string value containing a unique key for the current address.

Remarks

The Address Key is used by other Melissa Data WebSmart Services, such as GeoCoder and Delivery Indicator. The contents of this element is submitted with the Request Array of those web services.

Syntax

SOAP

```
string = Response.Record().Address.AddressKey
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <AddressKey>String</AddressKey>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Street Name

For each record in the Response Array, the web service returns a string value containing only the name portion of the street address.

Remarks

If the street address is “123 North Main Street,” this element would return “Main.”

If OptAddressParsed was not set to “True” by the Request Array, this element will not be populated.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.StreetName
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <StreetName>String</StreetName>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Address Range

For each record in the Response Array, the web service returns a string value containing only the street number portion of the street address.

Remarks

If the street address is “123 North Main Street,” this element would return “123.”

If OptAddressParsed was not set to “True” by the Request Array, this element will not be populated.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.AddressRange
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <AddressRange>String</AddressRange>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Address Suffix

For each record in the Response Array, the web service returns a string value containing only the suffix portion of the street address.

Remarks

If the street address is “123 North Main Street,” this element would return “St.”

If OptAddressParsed was not set to “True” by the Request Array, this element will not be populated.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.Suffix
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <Suffix>String</Suffix>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Address Direction

For each record in the Response Array, the web service returns a pair of string values containing only the directional portions of the street address, if any.

Remarks

If the street address is “123 North Main Street,” the Pre Direction element would return “N.” If the address is “456 Maple Street West,” the Post Direction element would return “W.”

If OptAddressParsed was not set to “True” by the Request Array, this element will not be populated.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.Direction.Post
string = Response.Record().Address.Parsed.Direction.Pre
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <Direction>
                    <Post>String</Post>
                    <Pre>String</Pre>
                </Direction>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Address Suite

For each record in the Response Array, the web service returns a pair of string values containing the parsed suite information from the street address.

Remarks

If the street address is “123 North Main Street #B,” the Suite Name element would return “Apt” and the Suite Number element would return “B.”

If OptAddressParsed was not set to “True” by the Request Array, this element will not be populated.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.Suite.Name
string = Response.Record().Address.Parsed.Suite.Range
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <Suite>
                    <Name>String</Name>
                    <Range>String</Range>
                </Suite>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Address PrivateMailbox

For each record in the Response Array, the web service returns a pair of string values containing the parsed private mailbox (PMB) information from a street address that belongs to a Commercial Mail Receiving Agency (CMRA).

Remarks

If OptAddressParsed was not set to “True” by the Request Array, this element will not be populated.

For more information on private mailboxes and how Address Verifier handles them, see *Secondary Addresses* on page 6.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.PrivateMailBox.Name
string = Response.Record().Address.Parsed.PrivateMailBox.Range
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <PrivateMailBox>
                    <Name>String</Name>
                    <Range>String</Range>
                </PrivateMailBox>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Route Service

For each Canadian address record in the Response Array, the web service returns the parsed route service information.

Remarks

Route service is a term used to refer to what the USPS would call a Rural Route address.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.RouteService
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <RouteService>String</RouteService>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Lockbox

For each Canadian address record in the Response Array, the web service returns the parsed lock box information.

Remarks

A lock box is the Canadian equivalent of a Post Office Box in the U.S. (The two terms are often used interchangeably in Canada).

Syntax

SOAP

```
string = Response.Record().Address.Parsed.LockBox
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <LockBox>String</LockBox>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Delivery Installation

For each Canadian address record in the Response Array, the web service returns the parsed delivery installation information.

Remarks

The delivery installation is the post office facility responsible for delivering to the current address. It is often used for rural addresses or when multiple post offices service the same municipality.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.DeliveryInstallation
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <DeliveryInstallation>
                    String
                </DeliveryInstallation>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Parsed Garbage

For each record in the Response Array, the web service returns a string value containing any portion of the street address that cannot be identified as a specific part.

Remarks

If the street address is “c/o 123 North Main Street,” this element would return “c/o.”

If OptAddressParsed was not set to “True” by the Request Array, this element will not be populated.

Syntax

SOAP

```
string = Response.Record().Address.Parsed.Garbage
```

XML

```
<ResponseArray>
    <Record>
        <Address>
            <Parsed>
                <Garbage>String</Garbage>
            </Parsed>
        </Address>
    </Record>
</ResponseArray>
```

Response Object XML Format

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

```
<?xml version="1.0" encoding="UTF-8"?>
<ResponseArray>
   <version>String</version>
   <TransmissionReference>String
   </TransmissionReference>
   <Results>String</Results>
   <TotalRecords>String</TotalRecords>

   <Record>
      <RecordID>String</RecordID>
      <Results>String</Results>
      <Address>
         <Company>String</Company>
         <Urbanization>
            <Name>String</Name>
         </Urbanization>
         <Address1>String</Address1>
         <Address2>String</Address2>
         <Suite>String</Suite>
         <PrivateMailBox>String</PrivateMailBox>
         <City>
            <Name>String</Name>
            <Abbreviation>String</Abbreviation>
         </City>
         <State>
            <Name>String</Name>
            <Abbreviation>String</Abbreviation>
         </State>
         <Zip>String</Zip>
         <Plus4>String</Plus4>
         <CarrierRoute>String</CarrierRoute>
         <DeliveryPointCode>String</DeliveryPointCode>
         <DeliveryPointCheckDigit>String
   </DeliveryPointCheckDigit>
         <Type>
            <Address>
               <Code>String</Code>
```

```
                <Description>String</Description>
            </Address>
            <Zip>
                <Code>String</Code>
                <Description>String</Description>
            </Zip>
        </Type>
        <Country>
            <Abbreviation>String</Abbreviation>
            <Name>String</Name>
        </Country>
        <AddressKey>String</AddressKey>
        <Parsed>
            <StreetName>String</StreetName>
            <AddressRange>String</AddressRange>
            <Suffix>String</Suffix>
            <Direction>
                <Post>String</Post>
                <Pre>String</Pre>
            </Direction>
            <Suite>
                <Range>String</Range>
                <Name>String</Name>
            </Suite>
            <PrivateMailbox>
                <Name>String</Name>
                <Range>String</Range>
            </PrivateMailbox>
            <RouteService>String</RouteService>
            <LockBox>String</LockBox>
            <DeliveryInstallation>
                String
            </DeliveryInstallation>
            <Garbage>String</Garbage>
        </Parsed>
    </Address>
</Record>
</ResponseArray>
```

Using Result Codes: Coding for the present and the future

Over a year ago, Melissa Data introduced a new concept know as Result codes. These are four-character codes (two letters followed by two numbers), delimited by commas, which indicate status and errors generated by the most recent request to an object or service. An Address Verifier result code for a coded address record might look something like this: “AC03, AC11, AS01, AS15.” Instead of looking at multiple properties and methods to determine the status or error of a record, you can simply look at the output of the Results property. Currently there are close to 50 possible result codes for Address Object alone. This section will dive into the best way to use these codes in your application now and in the future, focusing specifically on Address Verifier.

Best Practice #1: Read all the Result codes, but you won’t use them all.

The first step to understanding how to use Result codes is to know each code, individually. Having said that, understanding all the codes does not mean you will use all of them. You will likely only ever use a few. We have many different codes that indicate many different statuses or errors. A code may be important to one person but not another. For example, the AS20 codes means the address is deliverable only by the USPS, like a PO Box or a military address. This would not be important for you if you already delivery using USPS or don't deliver at all, but it would be important if you delivery using a third party carrier, like UPS.

Best Practice #2: Determine what a “good” record means.

The ultimate goal of using Result codes is to determine what to do with the record you have.

To do so, first determine what a “good” record is. In most cases, it will simply involve the AS01 or AS02 codes. For example, if you want your “good” record to be all addresses verified as fully deliverable, you would use:

```
if(Results.Contains("AS01")) { //good record}
```

If you want all fully deliverable addresses but also addresses that have missing/invalid suites, you would use:

```
if(Results.Contains("AS01") or Result.Contains("AS02")
   { //good record}
```

In more complex cases when you want to take more factors in account, add more code to your “good” record filter. For example, if you want all records that have a fully deliverable address or records that have an invalid suite but also a 10-digit verified phone number, you would write:

```
If(Results.Contains("AS01") or (Result.Contains("AS02")
   and Result.Contains("PS01"))
```

This filter introduces the Result codes for Phone Verifier, which behaves the same way as Address Verifier Result codes logically. Having said this, you can have more than one “good” filter. It is possible to cascade them in a “good,” “okay,” and then “bad”, in the same fashion as a switch statement. Once you have your “good” record filter, all the other records will naturally fall into the “bad” category.

Best Practice #3: Result codes will change. Code for it.

Since the inception of Result codes, the number of possible codes has doubled. Melissa Data is always innovating and adding new information and enrichments. You will not be able to know exactly what new codes may be introduced in the future, but we can still account for them. So, as we see in Best Practice #2, always use the `String.Contains()` or an equivalent function when detecting for codes, so re-ordering and future additions will not affect your current code. Also, have all records that do not pass your filter become a “bad” record. This allows for future codes to be added without records being lost if you don't specifically filter for them.

Like many things, the best way to learn how to use Result codes is to actually try and use them. See what Result codes are produced by different types of addresses, and how your code handles them. For an up-to-date online reference of all Result codes available and examples to produce each code, visit here: **http://www.melissadata.com/tech/ResultCodes.asp**.