Reference Guide

SmartMover™

Change of Address Processing

MELISSA DATA®

SmartMover
Web Service

Reference Guide

Melissa Data Corporation

Copyright

Trademarks

Melissa Data Corporation

22382 Avenida Empresa
Rancho Santa Margarita, CA 92688-2112

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

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

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

Document Code: DQTWSSMRG
Revision Number: 08072015.15

Dear Developer,

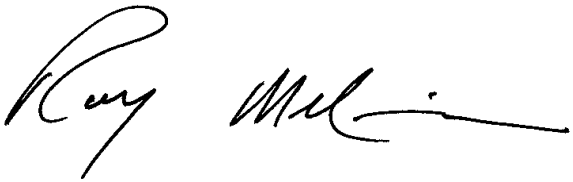
I would like to take this opportunity to thank you for your interest in Melissa Data products and introduce you to the company.

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

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

I look forward to hearing from you.

Best Wishes,



Raymond F. Melissa

President/CEO

Contents

SmartMover Web Service

Using the Service

Create a SmartMover Account

You must first have a Melissa Data sales representative create a SmartMover Account for you. In creating this account, you will need to determine and pay for credits based on the number of records you plan to process. You can call your sales representative and add additional credits to your account at any time but if you exceed your credit threshold, the SmartMover Web Service will stop processing and give you an error. When your account is created, you will receive an email with details about the service and your CustomerID number that you will use to process records.

Send a Processing Acknowledgement Form (PAF)

The USPS requires us to maintain a document for all customers using the SmartMover Web Service. Before you can start processing, you must submit a PAF to Melissa Data.

This can be done online at: http://www.melissadata.com/user/end_user_paf.aspx

For CCOA go here: https://www.melissadata.com/user/end_user_paf_cnd.aspx

Call the SmartMover Web Service

Once you have your CustomerID and your PAFID, you are ready to start processing. You start by building a SmartMover Request. This Request is made up of your CustomerID, PAFID, NCOA processing options, and an array of input records. You can fill this array with up to 100 records per call to the Request. Once you build your Request, you call the SmartMover Web Service and send us your Request. We will process your records and send back a Response with the results. Keep doing this until you finish processing your entire list.

Submitting/Retrieving Data

Service URLs

NCOA

SOAP	https://smartmover.melissadata.net/V3/SOAP/SmartMover
WEB/REST	https://smartmover.melissadata.net/V3/WEB/SmartMover

CCOA

SOAP	https://smartmovercanada.melissadata.net/V3/SOAP/SmartMover
WEB/REST	https://smartmovercanada.melissadata.net/V3/WEB/SmartMover

Add SmartMover WSDL to Your Project

If you are using 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 service URL on the Add Web Reference dialog box:

Name your web reference. For the sample code below, we will name it SmartMoverWS.

If you are not using Visual Studio.NET, see the documentation for your SOAP toolkit to see how to import the SmartMover wsdl.

Create SmartMover Web Service Objects

SmartMover Web Service uses the following objects to process your request and return you the results:

- Request: Array of Records to send to SmartMover
- RequestRecord: An individual record that make up the Request.
- SmartMover: Used to send and receive data
- Response: Array of Records received back from SmartMover
- ResponseRecord: An individual result record that make up the Response

Build your Request

First, fill in your CustomerID and PAFID information as well as what NCOA processing options you would like. Initialize the Request.Record variable as a new array. Then, for each individual input record, create a new RequestRecord and populate the input properties. Add that record into Request.Record until you run out of records or you reach 100 records.

Send the Request to SmartMover Web Service

After you have built your Request, call SmartMover.DoSmartMover(reqtArray) and pass in your Request as the parameter. This method will return you a Response as the result.

Gzip

The SmartMover web service supports the Gzip compression method available with many operating systems and programming languages. This will shrink the size of your tranmission data, lowering your bandwidth usage.

See your programming language documentation and gzip documentation for details.

Parse the Response

Once you get the Response back, parse it for the results and update your database. First, check Response.TransmissionResults to see if there was anything wrong with your Request or with the service itself. Now, go though the Response.Record array and process each individual ResponseRecord and put the results back into your database.

Sample Implementation Code

1. Create SmartMover Object:

```
SmartMoverWS.SmartMoverSOAPClient sm = new SmartMoverWS.
  SmartMoverSOAPClient();
```

2. Create A Request Object and fill in the NCOA processing information for this session:

```
SmartMoverWS.Request request = new SmartMoverWS.Request();
request.CustomerID = 1122334455;
request.ExecutionID = 1;
request.JobID = "ABCDEF"
request.PAFId = 12345;
request.Options = ProcessingType:Standard,
  ListOwnerFreqProcessingssing:1,NumberOfMonthsRequested:48;
request.Columns = grpParsed,grpStandardized;
```

3. Initialize the Request.Record array, maximum of 100:

```
request.Records = new SmartMoverWS.RequestRecord[100];
```

4. Loop through your records up to 100 at a time:

```
For count = 0 to 99 Do
```

5. For each record, create a new RequestRecord, fill it with your input record data, and add to the record array:

```
SmartMoverWS.RequestRecord reqRecord = new SmartMoverWS.
  RequestRecord();
reqRecord.RecordID = "Unique Identified"
reqRecord.Company = "Company Name if available"
reqRecord.NameFull = "Full Name if available"
reqRecord.AddressLine1 = "Address Line 1"
reqRecord.AddressLine2 = "Address Line 2 if present"
reqRecord.City = "City"
reqRecord.State = "State"
reqRecord.PostalCode = "Postal Code"
request.Records[count] = reqRecord
```

6. After you have added all your records to the Record array, set TotalRequests to the number of records you added:

```
reqquest.TotalRecords = count + 1;
```

7. Send the Request to the SmartMover Web Service and get a Response back:

```
SmartMoverWS.Response response = sm.DoSmartMover(request);
```

8. Check the Response for any faults:

```
if (response.Fault.Code != "")
{
  //Handle error
}
```

9. Loop through the Record array and retrieve all record results:

```
For count = 0 to (response.TotalRecords - 1) Do
```

10. For each ResponseRecord inside the Record array, check the Status code to see if the record was moved, standardized, or an error:

```
if (response.Records[x].Results.Contains("AE"))
{
  //There was an error in the input address
  String ErrorCode = response.Records[x].Results;
}
if (response.Records[x].Results.Contains("CS01"))
{
  //This record was a move and the new address is returned
  String NewName = response.Records[x].NameFull;
  String NewCompany = response.Records[x].CompanyName;
  String NewAddressLine1 = response.Records[x].AddressLine1;
  String NewSuite = response.Records[x].Suite;
  String NewCity = response.Records[x].City;
  String NewState = response.Records[x].StateName;
  String NewPostalCode = response.Records[x].PostalCode;
  String NewPlus4 = response.Records[x].Plus4;
  String NewAddressLine1 = response.Records[x].AddressLine1;
}
else
{
  //This record did not move, but has been verified, standardized,
  and/or corrected
  String StdName = response.Records[x].NameFull;
  String StdCompany = response.Records[x].CompanyName;
  String StdAddressLine1 = response.Records[x].AddressLine1;
  String StdSuite = response.Records[x].Suite;
  String StdCity = response.Records[x].City;
```

```
    String StdState = response.Records[x].StateName;
    String StdPostalCode = response.Records[x].PostalCode;
    String StdPlus4 = response.Records[x].Plus4;
    String StdAddressLine1 = response.Records[x].AddressLine1;
  }
```

Request

The Request object contains information from the user to be processed by the SmartMover Web Service. This includes the fields to identify the customer to the service and set the desired NCOA[Link] options, in addition to an array of address records to be processed.

Customer Information and Option Fields

The following fields identify the user to the SmartMover Web Service and set the necessary options to process a list of address records. Except where indicated, all of these fields should be sent as string values.

Actions

```
XML/SOAP: Request.Actions
JSON: Actions
REST: act
```

Actions is used to select which COA object(s) will be used in this request. Actions are comma delimited when more than one is selected.

For a list of actions, please see “Actions” on page 36.

Columns

```
XML/SOAP: Request.Columns
JSON: Columns
REST: cols
```

Columns allow you to add more fields to the response in additions to the default fields. These can be empty, have a single group name, or a combination of group names. If a combination is used, comma “,” delimit the group names with no spaces. For example:

```
grpParsed,grpName,grpOriginal,grpStandardized
```

For a list of default columns and column groups, please see “Column Groups” on page 37.

Options

```
XML/SOAP: Request.Options
JSON: Options
REST: opt
```

If an option's value is not set, the default value will be used. Options should be comma "," delimited with no spaces and use a colon ":" to separate the option from its value. For example:

```
ProcessingType:Standard,ListOwnerFreqProcessing:1
```

Not applicable to CCOA.

For a list of options and their values, please see "Options" on page 36.

ProcessingType

```
ProcessingType
```

This field accepts an option from the ProcessingType enumeration. For a list of available options, please see "ProcessingType Descriptions" on page 37.

ListOwnerFreqProcessing

```
ListOwnerFreqProcessing
```

This field accepts an integer value from 1 to 52. This is the number of times per year that the current mailing list is used for mailing. If you use it monthly, enter 12; for quarterly, use 4; etc.

NumberofMonthsRequested

```
NumberofMonthsRequested
```

The field accepts an integer value from 6 to 48. This is the number of months back that you want the web service to search for a change of address.

CustomerID

```
XML/SOAP: Request.CustomerID
JSON: CustomerID
REST: id
```

The customerID is a string of characters issued by Melissa Data when you open your SmartMover Web Service account.

PAFId

```
XML/SOAP: Request.PAFID
JSON: PAFID
REST: pafid
```

The PAF is a string value which identifies the individual list owner when the request is submitted from a broker account.

This value is required for any CustomerID issued to a broker account. Not applicable to CCOA.

JobID

```
XML/SOAP: Request.JobID
JSON: JobID
REST: jobid
```

The JobID is a unique string of characters used to identify a group records submitted with this request. The JobID is returned with the Response Array, making it simple to connect the records returned with the original request.

This JobID is also used to retrieve the various summary reports returned by the Smart Mover Web Service.

ExecutionID

```
XML/SOAP: Request.ExecutionID
JSON: ExecutionID
```

The ExecutionID identifies the thread that you are processing under when using multiple threads. This allows our system to track individual threads in the event that a request times out. If you send the same request immediately after a request times out, using the same ExecutionID, our system will process the 2nd request but it will not count against the total number of records in the NCOA and CASS reports. The default value is 0 if not set.

OptSmartMoverListName

```
SOAP/XML: Request.OptSmartMoverListName
JSON: OptSmartMoverListName
REST: List
```

This is the name that identifies the current list. It will be included in the reports that the SmartMover Web Service returns after processing.

Request Record Fields

Request.Record is an array of records containing the addresses to be processed.

Each record within the Request contains some or all of the following fields. The following fields are required for each record:

1. Either NameFirst and NameLast, NameFull or Company
2. Address
3. Either City and State, ZIP or AddressLastLine

RecordID

```
XML/SOAP: Request.Records[index].RecordID
JSON: Records[index].RecordID
```

This is a unique identifier for this record from your own database. This is not required but it can assist in matching the record in the Response with the original record.

Company

```
XML/SOAP: Request.Records[index].Company
JSON: Records[index].Company
REST: comp
```

If this address is that of a business, include the company name here.

Urbanization

```
XML/SOAP: Request.Records[index].Urbanization
JSON: Records[index].Urbanization
REST: u
```

This field is only used for addresses located in Puerto Rico and is used to break ties between similar addresses in the same Postal Code.

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

AddressLine1

```
XML/SOAP: Request.Records[index].AddressLine1
JSON: Records[index].AddressLine1
REST: a1
```

This is the primary street address. It may also include the suite number.

AddressLine2

```
XML/SOAP: Request.Records[index].AddressLine2
JSON: Records[index].AddressLine2
REST: a2
```

This may include the suite or mailbox number or an alternate address, such as a P.O. Box.

Suite

```
XML/SOAP: Request.Records[index].Suite
JSON: Records[index].Suite
REST: ste
```

If your address stores the suite number separately, pass it to the web service via this field.

PrivateMailbox

```
XML/SOAP: Request.Records[index].PrivateMailbox
JSON: Records[index].PrivateMailbox
REST: pmb
```

If this address is actually a box in a private mailbox service, and the number is stored separately enter the number here.

City

```
XML/SOAP: Request.Records[index].City
JSON: Records[index].City
REST: city
```

Pass the full name of the city via this field.

State

```
XML/SOAP: Request.Records[index].State
JSON: Records[index].State
REST: state
```

Pass either the full name or the two-letter abbreviation for the state via this field.

PostalCode

```
XML/SOAP: Request.Records[index].PostalCode
JSON: Records[index].PostalCode
REST: postalcode
```

This could either be a five-digit ZIP Code, the first five digits of a ZIP + 4 or the full nine-digit ZIP + 4.

Plus4

```
XML/SOAP: Request.Records[index].Plus4
JSON: Records[index].Plus4
REST: plus4
```

Use this for the last four digits of a ZIP + 4 if not included with Zip field.

Country

```
XML/SOAP: Request.Records[index].Country
JSON: Records[index].Country
REST: ctry
```

You may pass a country code via this field, but the SmartMover Web Service can only update addresses within the United States.

NameFull

```
XML/SOAP: Request.Records[index].NameFull
JSON: Records[index].NameFull
REST: full
```

If this is an individual person's address and the entire name is stored as a single string, pass the string value via this field. The name will be parsed when the record is returned via the Response.

NameFirst

```
XML/SOAP: Request.Records[index].NameFirst
JSON: Records[index].NameFirst
REST: first
```

If this is an individual person's address and the first name is stored separately, pass the string value via this field.

NameMiddle

```
XML/SOAP: Request.Records[index].NameMiddle
JSON: Records[index].NameMiddle
REST: middle
```

If this is an individual person's address and the middle name is stored separately, pass the string value via this field.

NameLast

```
XML/SOAP: Request.Records[index].NameLast
JSON: Records[index].NameLast
REST: last
```

If this is an individual person's address and the last name is stored separately, pass the string value via this field.

NamePrefix

```
XML/SOAP: Request.Records[index].NamePrefix
JSON: Records[index].NamePrefix
REST: namepre
```

This would be a title or honorific such as "Mr," "Miss," or "Dr."

NameSuffix

```
XML/SOAP: Request.Records[index].NameSuffix
JSON: Records[index].NameSuffix
REST: namesfx
```

This would be a generational or professional suffix, such as "Jr," "IV" or "Ph.D."

Response

The SmartMover Web Service returns its results in the Response Object. This object contains any fault codes generated by the request, links to summary reports, and an array containing the results for each record.

Response Details

TotalRecords

```
XML/SOAP: Response.TotalRecords
JSON: TotalRecords
```

This property returns the total number of records processed. It should match the TotalRecords field in the original request.

JobID

```
XML/SOAP: Response.JobID
JSON: JobID
```

This field returns the JobID submitted with the request array, making it simple to matched the records returned with the original request.

TransmissionResults

```
XML/SOAP: Response.TransmissionResults
JSON: TransmissionResults
```

If the request failed this field will contain a code that describes the nature of the failure. For a list of possible codes returned for this field, please see “SmartMover Result Codes” on page 27.

Record

```
XML/SOAP: Response.Records[index]
JSON: Records[index]
```

The Response returns an array of records, one for each submitted, containing the processed address information and the name or company information.

RecordID

```
XML/SOAP: Response.Records[index].RecordID
JSON: Records[index].RecordID
```

This returns the unique identifier submitted with the original request. Use this field to match the current record with the corresponding record in your own database.

CompanyName

```
XML/SOAP: Response.Records[index].CompanyName
JSON: Records[index].CompanyName
```

Returns a string value containing the company name for this record as passed via the Request.

Results

```
XML/SOAP: Response.Records[index].Results
JSON: Records[index].Results
```

This field returnd a comma-delimted set of four-character codes indicating the address verification and move status of the current record.

For a list of result codes returned, please see “Result Codes” on page 27.

AddressTypeCode

Most addresses have a type code associated with them. Some postal codes also have a type code which indicate that the postal code has a special purpose, such as being assigned to a military facility.

```
XML/SOAP: Response.Records[index].AddressTypeCode
JSON: Records[index].AddressTypeCode
```

These fields return the address type code and the accompanying description. For a list of possible codes, please see .”Address Type Codes” on page 36

Urbanization

```
XML/SOAP: Response.Records[index].Urbanization
JSON: Records[index].Urbanization
```

This field returns the value passed to the Urbanization field in the Request Array. For more information, please see ”Urbanization” on page 10.

AddressLine1

```
XML/SOAP: Response.Records[index].AddressLine1
JSON: Records[index].AddressLine1
```

This is the standardized or updated version of the street address in the Address field of the Request. This may include the contents of AddressLine2 if the Address Check process had to swap the two fields to verify the address. The suite can be returned as part of this field unless the suite field has been requested in which case the suite will be returned there.

AddressLine2

```
XML/SOAP: Response.Records[index].AddressLine2
JSON: Records[index].AddressLine2
```

This is the standardized or updated version of the street address in the AddressLine2 field of the Request. This may include the contents of AddressLine1 if the Address Check process had to swap the two fields to verify the address. The suite can be returned as part of this field unless the suite field has been requested in which case the suite will be returned there.

Suite

```
XML/SOAP: Response.Records[index].Suite
JSON: Records[index].Suite
```

This field may have been populated by suite information from the AddressLine1 or AddressLine2 fields of the corresponding record in the Request.

PrivateMailbox

```
XML/SOAP: Response.Records[index].PrivateMailbox
JSON: Records[index].PrivateMailbox
```

This field returns the private mail box number associated with a CMRA (Commercial Mail Receiving Agency).

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.

City

```
XML/SOAP: Response.Records[index].City
JSON: Records[index].City
```

This field contains the city name. If the city name is more than 28 characters long, the Abbreviation will contain the official USPS abbreviation for that city.

CityAbbreviation

```
XML/SOAP: Response.Records[index].CityAbbreviation
JSON: Records[index].CityAbbreviation
```

This field contains the city abbreviation. If the city name is more than 28 characters long, the Abbreviation will contain the official USPS abbreviation for that city.

State

```
XML/SOAP: Response.Records[index].State
JSON: Records[index].State
```

This field returns the standard two-letter abbreviation of the state in the returned address.

StateName

```
XML/SOAP: Response.Records[index].StateName
JSON: Records[index].StateName
```

This field returns the full name of the state in the returned address.

PostalCode

```
XML/SOAP: Response.Records[index].PostalCode
JSON: Records[index].PostalCode
```

This will contain only the five-digit ZIP Code, even if the full ZIP + 4 was passed via the Zip field in the Request.

Plus4

```
XML/SOAP: Response.Records[index].Plus4
JSON: Records[index].Plus4
```

This field returns the last four digits of a nine-digit ZIP + 4 code.

CarrierRoute

```
XML/SOAP: Response.Records[index].CarrierRoute
JSON: Records[index].CarrierRoute
```

The first character of this property is always alphabetic, and the last three characters are numeric. For example, "R001" or "C027" would be typical carrier routes. The alphabetic letter indicates the type of delivery associated with this address. For a list of possible codes, please see "Carrier Route Codes" on page 35.

DeliveryPointCode

```
XML/SOAP: Response.Records[index].DeliveryPointCode
JSON: Records[index].DeliveryPointCode
```

The delivery point code contains the 10th and 11th digits of the POSTNET barcode, usually the last two digits of the street number.

DeliveryPointCheckDigit

```
XML/SOAP: Response.Records[index].DeliveryPointCheckDigit
JSON: Records[index].DeliveryPointCheckDigit
```

The delivery point code contains the 12th digit of the POSTNET barcode.

DPVFootnotes

DPV enables the Post Office to validate that a submitted address actually corresponds to a real, deliverable address. DPV processing is now required for CASS Certified address checking software like Melissa Data's MAILERS+4 and Address Object.

```
XML/SOAP: Response.Records[index].DPVFootnotes
JSON: Records[index].DPVFootnotes
```

The DPV footnotes indicate the level of matching between the current address and the USPS's DPV database. The footnote may be up to six characters long. For a list of possible codes, please see "DPV Address Status Codes" on page 35.

CountryName

```
XML/SOAP: Response.Records[index].CountryName
JSON: Records[index].CountryName
```

Returns the full name of the country in which the address is located. Currently, only Canada and the United States are supported.

CountryCode

```
XML/SOAP: Response.Records[index].CountryCode
JSON: Records[index].CountryCode
```

Returns the standard code of the country in which the address is located. Currently, only Canada and the United States are supported.

AddressKey

```
XML/SOAP: Response.Records[index].AddressKey
JSON: Records[index].AddressKey
```

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

MoveTypeCode

```
XML/SOAP: Response.Records[index].MoveTypeCode
JSON: Records[index].MoveTypeCode
```

The Move Type indicates the type of address record that was matched. For a list of possible codes, please see “Move Type Codes” on page 35.

MoveReturnCode

```
XML/SOAP: Response.Records[index].MoveReturnCode
JSON: Records[index].MoveReturnCode
```

The return code indicates the level of matching between the current record and the NCOA^{Link} database. For a list of possible codes, please see “NCOA Return Codes” on page 32.

MoveEffectiveDate

```
XML/SOAP: Response.Records[index].MoveEffectiveDate
JSON: Records[index].MoveEffectiveDate
```

US Only. This field returns the effective date of the move as a string value in the format “YYYYMM.”

NameFull

```
XML/SOAP: Response.Records[index].NameFull
JSON: Records[index].NameFull
```

Returns the full name as submitted to the NameFull field of a record in the Request.

NameFirst

```
XML/SOAP: Response.Records[index].NameFirst
JSON: Records[index].NameFirst
```

Returns the first name as submitted to the NameFirst field of a record in the Request.

NameMiddle

```
XML/SOAP: Response.Records[index].NameMiddle
JSON: Records[index].NameMiddle
```

Returns the middle name as submitted to the NameMiddle field of a record in the Request.

NameLast

```
XML/SOAP: Response.Records[index].NameLast
JSON: Records[index].NameLast
```

Returns the last name as submitted to the NameLast field of a record in the Request.

NamePrefix

```
XML/SOAP: Response.Records[index].NamePrefix
JSON: Records[index].NamePrefix
```

Returns the name prefix as submitted to the NamePrefix field of a record in the Request.

NameSuffix

```
XML/SOAP: Response.Records[index].NameSuffix
JSON: Records[index].NameSuffix
```

Returns the name suffix as submitted to the NameSuffix field of a record in the Request.

Parsed Address Fields

If grpParsed was selected when the Request was submitted, the Response will return the following fields.

AddressStreetName

```
XML/SOAP: Response.Records[index].AddressStreetName
JSON: Records[index].AddressStreetName
```

Returns just the name portion of the street address, minus the street number, suffix and any directionals.

AddressHouseNumber

```
XML/SOAP: Response.Records[index].AddressHouseNumber
JSON: Records[index].AddressHouseNumber
```

Returns just the numeric portion of the street address as a string value.

AddressStreetSuffix

```
XML/SOAP: Response.Records[index].AddressStreetSuffix
JSON: Records[index].AddressStreetSuffix
```

Returns the street name suffix, such as “Rd,” St,” “Blvd,” and so on.

AddressPreDirection

```
XML/SOAP: Response.Records[index].AddressPreDirection
JSON: Records[index].AddressPreDirection
```

Returns any directional abbreviation that comes before the street name. “100 North Main Street” would return “N.”

AddressPostDirection

```
XML/SOAP: Response.Records[index].AddressPostDirection
JSON: Records[index].AddressPostDirection
```

Returns any directional abbreviation that comes after the street name. An address on Park Ave South would return “S.”

AddressSuiteName

```
XML/SOAP: Response.Records[index].AddressSuiteName
JSON: Records[index].AddressSuiteName
```

Returns standardized text, such as “STE” or “Unit,” that is part of the suite number.

AddressSuiteNumber

```
XML/SOAP: Response.Records[index].AddressSuiteNumber
JSON: Records[index].AddressSuiteNumber
```

Returns only the numeric portion of suite number as a string value.

AddressPrivateMailboxName

```
XML/SOAP: Response.Records[index].AddressPrivateMailboxName
JSON: Records[index].AddressPrivateMailboxName
```

This field returns the non-numeric portion of a private mailbox number, either “#” or “PMB”

AddressPrivateMailboxRange

```
XML/SOAP: Response.Records[index].AddressPrivateMailboxRange
JSON: Records[index].AddressPrivateMailboxRange
```

If the address is located at a CMRA (Commercial Mail Receiving Agency), the numeric portion is returned here as a string value.

AddressRouteService

```
XML/SOAP: Response.Records[index].AddressRouteService
JSON: Records[index].AddressRouteService
```

Returns the route service.

AddressLockBox

```
XML/SOAP: Response.Records[index].AddressLockBox
JSON: Records[index].AddressLockBox
```

Returns the lockbox.

AddressExtras

```
XML/SOAP: Response.Records[index].AddressExtras
JSON: Records[index].AddressExtras
```

This field returns any text that could not be identified as belonging in any of the above fields.

Original

The original address is returned exactly as passed to the Request, so the Original address includes those fields, including the LastLine field. This group of fields is activated using grpOriginal in the columns field.

```
XML/SOAP:
Response.Records[index].OriginalUrbanization
Response.Records[index].OriginalAddressLine1
Response.Records[index].OriginalAddressLine2
Response.Records[index].OriginalSuite
Response.Records[index].OriginalPrivateMailbox
Response.Records[index].OriginalCity
Response.Records[index].OriginalCityAbbreviation
Response.Records[index].OriginalState
Response.Records[index].OriginalStateName
Response.Records[index].OriginalPostalCode
Response.Records[index].OriginalPlus4
Response.Records[index].OriginalCountryCode
Response.Records[index].OriginalResults
```

Field names for JSON are the same, only without “Response.”. For example:

```
JSON:
Records[index].OriginalUrbanization
```

Standardized

The Standardized address is the original address, DPV coded and standardized using USPS addressing standards. This group of fields is activated using grpStandardized in the columns field.

```
...StandardizedResults
...StandardizedUrbanization
...StandardizedAddressLine1
...StandardizedAddressLine2
...StandardizedSuite
...StandardizedPrivateMailbox
```

```
...StandardizedCity
...StandardizedCityAbbreviation
...StandardizedState
...StandardizedStateName
...StandardizedPostalCode
...StandardizedPlus4
...StandardizedCarrierRoute
...StandardizedDeliveryPointCode
...StandardizedDeliveryPointCheckDigit
...StandardizedCountryCode
```

Get Summary Report Link Service

The Get Summary Report Link Service will return URL links to web pages that contain the CASS 3553 and NCOA^{Link} forms based on the customer ID and JobID.

This applies only to NCOA, not CCOA.

Using the Service

Getting a Summary Report Link from the SmartMover is simple process.

1. Create SmartMover Object:

   ```
   SmartMoverWS.SmartMover sm = new SmartMoverWS.SmartMover();
   ```

2. Create a RespNCOASummaryReport Object and call the GetSummaryReportLink method, passing the customer ID number and JobID as shown.

   ```
   SmartMoverWS.RespNCOALinkReportLink SummaryLinks =
     sm.GetSummaryReportLink (customerID, JobID);
   ```

Appendix

Using Result Codes

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 Object 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 Object.

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 Object, which behaves the same way as Address Object 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 the Melissa Data website.

Result Codes

SmartMover Result Codes

Code	Short Description	Long Description
SE00	Unexpected Error	Your request could not be processed due to an unexpected error. Please retry your request.
SE10	Record Number Mismatch	Total Records numbers do not match with the records array.
SE11	Month Range Exceeded	Months to allow for NCOALink products ranges from 6 to 48.
SE12	Records Array Error	Records array is empty, un-allocated, or out of bounds.
SE13	Requesting Record Range Exceeded	Requesting records must in the range of 1 to 100.
SE14	No Summary Reports	Summary reports do not exist.
SE15	ProcessingType Error	ProcessingType is empty or incorrect type.

Code	Short Description	Long Description
SE16	FreqProcessing Range Exceeded	ListOwnerFreqProcessing ranges from 1 to 52.
SE17	ListName Character Limit	OptSmartMoverListName cannot contain over 30 characters.
SE18	ListName Invalid Character	OptSmartMoverListName cannot contain any of the following characters \/*:?""<>| or non-printing characters.
SE19	Invalid Link	Invalid link.
SE20	CustomerID Error	CustomerID is not valid or customer's Paf is not on file.
SE21	SmartMover Service Disabled	Smart Mover Service package(s) is/are not enabled. Please contact your sales representative!
SE22	Long JobID	JobID cannot contain over 50 characters.
SE23	JobID Invalid Character	JobID cannot contain any of the following characters \/*:?""<> |, non-printing characters or spaces.
SE24	No JobID	JobID required to open summary reports.
SE25	Customer PAF Expired	The Customer PAF is expired.
SE26	Broker PAF Expired	The Broker PAF is expired.
SE27	NO PAFID Match	The PAFID does not match.
SE28	No PAFID	The under broker customer required a PAFID.
SE29	Unconfirmed PAFID	The PAFID is unconfirmed.
SE30	Empty XML	Incorrect or empty XML/JSON structure.
SE31	Empty Record Structure	Incorrect or empty record structure.
SE32	Incorrect Action	The attempted action(s) was incorrect.

Address Check Result Codes

Code	Short Description	Long Description
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.
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.
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.
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.

Code	Short Description	Long Description
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.
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.
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.

Move Result Codes

Code	Short Description	Long Description
CS01	Move with New Address	The record is a 'move' and a new 'moved to' address was provided.

Code	Short Description	Long Description
CS02	Standardized Address	The record is not a 'move' but was standardized.
CS03	Move Input Requirements not Satisfied	The record is matched to change of address file but did not satisfy all requirements to produce a 'moved to' address.
CS04	Move but No New Address	The record is a 'move' but could not provide a 'moved to' address.
CS10	Individual Move	The record is classified as an individual.
CS11	Family Move	The record is classified as a family.
CS12	Business Move	The record is classified as a business.
CS13	Daily Delete	The record is a Daily Delete address. COA with this address is pending deletion from the master file and no mail may be forwarded from this address.
CM01	COA Match	A COA was found for an Individual, Business, or Family.
CM02	Foreign Move	A COA was found for a Foreign Move. No forwarding address was provided.
CM03	Moved no Forwarding	A COA was found. The customer moved and left no forwarding address (MLNA).
CM04	Box Closed	A COA was found. The Post Box was closed with no order filed by the customer (BCNO).
CM05	Cannot Match Secondary	The COA could not be matched because the street address does not match with the secondary number.
CM06	DPBC Ambiguous	The COA could not be converted to a deliverable address because the 11-digit zip code represents more than one delivery point.
CM07	Conflicting Middle Name	The COA could not be matched because there is more than one COA record for the match and the middle names or initials on the COAs are different.
CM08	Conflicting Gender	The COA could not be matched because there is more than one COA record for the match and the genders of the names on the COAs are different.
CM09	Conflicting Instructions	The COA could not be matched because two potential matches were compared and had differences in the new addresses.
CM10	Cannot Match High-rise Default	The COA could not be matched because the input record is a potential match to a family COA record from a multi-unit building, but the address zip + 4 coded to the building default. This requires individual name matching logic to obtain a match and the individual names did not match.
CM11	Cannot Match Rural Default	The COA could not be matched because the input record is a potential match to a family COA record from a Rural Route or Highway Contract Route address. This requires individual name matching logic to obtain a match and the individual names did not match.

Code	Short Description	Long Description
CM12	Insufficient Name	The COA could not be matched because there is a COA record with the same surname and address but there is insufficient first/middle name information on the COA record to product a match using individual matching logic.
CM13	Middle Name Test Failed	The COA could not be matched because the input name contains a conflict with the middle name or initials on the COA record.
CM14	Gender Conflict	The COA could not be matched because the gender of the name on the input record conflicts with the gender of the name on the COA record.
CM15	Cannot Convert Address	The COA was found but the new address would not convert at run time.
CM16	Individual Name Insufficient	The COA could not be matched because the input middle initial/name is missing or does not equal the middle initial/name on the potential COA match.
CM17	Secondary Descrepancy	The COA could not be matched because there is conflicting secondary information on the input and COA record or the input record contained secondary information and matched to a family record that does not contain secondary information.
CM18	Other Insufficient Name	The COA could not be matched because the input middle initial/name is missing or different from the middle name on the COA. A match cannot be made because the first name on the COA was truncated and the middle names must be equal in order to make this match.
CM19	Cannot Match General Delivery	The COA could not be matched for general delivery.
CM20	No ZIP + 4, Not Confirmable, or Temporary	The COA was found but the new address either: does not have a ZIP+4 coded and therefore there is no 11-digit DPBC, the primary number cannot be confirmed on DPV, or is temporary.
CM21	Conflicting Directions	The COA could not be matched due to conflicting directions after rechaining.
CM22	Secondary Dropped from COA	The secondary number was dropped from COA.
CM23	Secondary Dropped from Input	The secondary number was dropped from the input address.

NCOA Return Codes

Code	Short Definition	Long Definition
A	COA Match	The input record matched to a business, individual or family type master file record. A new address could be furnished.
00	No Match	The input record COULD NOT BE matched to a master file record. A new address could not be furnished.

Code	Short Definition	Long Definition
01	Foreign Move	The input record matched to a business, individual or family type master file record but the new address was outside USPS delivery area.
02	Moved Left No Address	The input record matched to a business, individual or family type master file record and the new address was not provided to USPS.
03	Box Closed No Order (BCNO)	The Input record matched to a business, individual or family type master file record which contains an old address of PO BOX that has been closed without a forwarding address provided.
04	Street Address with Secondary	In the STANDARD mode utilizing Family matching logic the input record matched to a family record type on the master file with an old address that contained secondary information which obtained a ZIP+4 street level match. The input record does not contain secondary information. This address match situation requires individual name matching logic to obtain a match and individual names do not match.
05	New 11-digit DPBC is Ambiguous	The input record matched to a business, individual or family type master file record. The new address on the master file record could not be converted to a deliverable address because the DPBC represents more than one delivery point.
06	Conflicting Directions: Middle Name Related	There is more than one COA (individual or family type) record for the match algorithm and the middle names or initials on the COAs are different. Therefore, a single match result could not be determined.
07	Conflicting Directions: Gender Related	There is more than one COA (individual or family type) record for the match algorithm and the genders of the names on the COAs are different. Therefore, a single match result could not be determined.
08	Other Conflicting Instructions	The input record matched to two master file (business, individual or family type) records. The two records in the master file were compared and due to differences in the new addresses, a match could not be made.
09	High-rise Default	The input record matched to a family record on the master file from a High-rise address ZIP+4 coded to the building default. This address match situation requires individual name matching logic to obtain a match and individual names do not match.
10	Rural Default	The input record matched to a family record on the master file from a Rural Route or Highway Contract Route address ZIP+4 coded to the route default. This address situation requires individual name matching logic to obtain a match and individual names do not match.
11	Insufficient COA Name for Match	There is a master file (individual or family type) record with the same surname and address but there is insufficient name information on the master file record to produce a match using individual matching logic.
12	Middle Name Test Failed	The input record matched to an individual or family record on the master file with the same address and surname. However, a match cannot be made because the input name contains a conflict with the middle name or initials on the master file record.

Code	Short Definition	Long Definition
13	Gender Test Failed	The input record matched to a master file (individual or family type) record. A match cannot be made because the gender of the name on the input record conflicts with the gender of the name on the master file record.
14	New Address Would Not Convert	The input record matched to a master file (business, individual or family type) record. The new address could not be converted to a deliverable address.
15	Individual Name Insufficient	There is a master file record with the same address and surname. A match cannot be made because the input record does not contain a first name or contains initials only.
16	Secondary Number Discrepancy	The input record matched to a street level individual or family type record. However, a match is prohibited based on I of the following reasons: 1) There is conflicting secondary information on the input and master file record; 2) the input record contained secondary information and matched to a family record that does not contain secondary information. In item 2, this address match situation requires individual name matching logic to obtain a COA match and individual names do not match.
17	Other Insufficient Name	The input record matched to an individual or family master file record. The input name is different or not sufficient enough to produce a match.
18	General Delivery	The input record matched to a family record on the master file from a General Delivery address. This address situation requires individual name matching logic to obtain a match and individual names do not match.
19	New Address not ZIP + 4 coded or DPV confirm.	There is a change of address on file but the new address cannot be ZIP + 4 coded and therefore there is no 11-digit DPBC to store or return, the new address primary number cannot be confirmed on DPV or the new address is temporary.
20	Conflicting Directions after re-chaining	Multiple master file records were potential matches for the input record. The master file records contained different new addresses and a single match result could not be determined.
66	Daily Delete	The input record matched to a business, individual or family type master file record with an old address that is present in the daily delete file. The presence of an address in the daily delete file means that a COA with this address is pending deletion from the master file and that no mail may be forwarded from this address.
77	ANK^{Link} Primary Return Code	Indicates that ANKlink has provided the move date, but not address, for a move that occurred between 19 and 48 months ago.
91	Secondary Number dropped from COA	The input record matched to a master file record. The master file record had a secondary number and the input address did not. Please Note: This return code is derived from Individual matching logic only. If this return code is achieved then no other matching attempts are permitted regardless of the PROCESSING mode.

Code	Short Definition	Long Definition
92	Secondary Number Dropped from input address	The input record matched to a master file record, but the input address had a secondary number and the master file record did not. The record is a ZIP + 4 street level match. Please Note: This return code is derived from individual matching logic only. If this return code is achieved then no other matching attempts are permitted regardless of the PROCESSING mode.

Move Type Codes

Code	Description
F	Family
B	Business
I	Individual

DPV Address Status Codes

Code	Description
AA	Input Address Matched to the ZIP + 4 file
A1	Input Address Not Matched to the ZIP + 4 file
BB	DPV matched (all components)
CC	Primary Number Match – Secondary present but invalid
F1	Address Was Coded to a Military Address
G1	Address Was Coded to a General Delivery Address
M1	Primary Number missing
M3	Primary Number invalid
N1	Primary Number Match – Secondary missing
P1	Missing PO, RR or HC Box number
P3	Invalid PO, RR or HC Box number
R1	DPV matched to CMRA – PMB number not present
RR	DPV matched to CMRA
U1	Address Was Coded to a Unique ZIP Code

Carrier Route Codes

Code	Description
B	PO Box

Code	Description
C	City Delivery
G	General Delivery
H	Highway Contract
R	Rural Route

Address Type Codes

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

Actions

Code	Description
NCOA	US Change of Address Object
CCOA	Canadian Change of Address Object

Options

Option	Default Value	Possible Values	Description
ProcessingType	Standard	**Standard**	The processing type. For a more detailed description, please see the following table.
		Individual	
		IndividualAndBusiness	
		Business	
		Residential	
ListOwnerFreqProcessing	1	**1** to 52	The frequency of mailing in number of weeks.
NumberOfMonthsRequested	48	6 to **48**	The number of months of service.

ProcessingType Descriptions

Value	Option
Standard	Standard Processing Mode requires inquiries in the following order: • Business – Match on business name. • Individual – Match on first name, middle name, surname and title required. Gender is checked and nickname possibilities are considered. • Family – Match on surname only.
IndividualAndBusiness	The NCOALink customer may choose to omit all "Family" match inquiries and allow only "Individual" and "Business" matches to be acceptable.
Individual	The NCOALink customer may also choose to omit "Business" match inquiries when processing individual names for mailing lists that contain no business addresses
Business	The NCOALink customer may choose to process for only "Business" matches when processing a "Business-to-Business" mailing list which contains no residential (Individual or Family) addresses.
Residential	The NCOALink customer may choose to omit "Business" match inquiries and allow only "Individual" and "Family" matches to be acceptable under Residential Processing Mode.

Column Groups

Column Name	Default	Group Name
<AddressDeliveryInstallation />		grpParsed
<AddressExtras />	Default	
<AddressHouseNumber />		grpParsed
<AddressKey />	Default	
<AddressLine1 />	Default	
<AddressLine2 />	Default	
<AddressLockBox />		grpParsed
<AddressPostDirection />		grpParsed
<AddressPreDirection />		grpParsed
<AddressPrivateMailboxName />		grpParsed
<AddressPrivateMailboxRange />		grpParsed
<AddressRouteService />		grpParsed
<AddressStreetName />		grpParsed
<AddressStreetSuffix />		grpParsed
<AddressSuiteName />		grpParsed

Column Name	Default	Group Name
<AddressSuiteNumber />		grpParsed
<AddressTypeCode />	Default	
<CarrierRoute />	Default	
<City />	Default	
<CityAbbreviation />	Default	
<CompanyName />	Default	
<CountryCode />	Default	
<CountryName />	Default	
<DeliveryPointCheckDigit />	Default	
<DeliveryPointCode />	Default	
<DPVFootNotes />		ColumnOnly
<MoveEffectiveDate />	Default	
<MoveReturnCode />		ColumnOnly
<MoveTypeCode />	Default	
<NameFirst />		grpName
<NameFull />		grpName
<NameLast />		grpName
<NameMiddle />		grpName
<NamePrefix />		grpName
<NameSuffix />		grpName
<OriginalAddressLine1 />		grpOriginal
<OriginalAddressLine2 />		grpOriginal
<OriginalCity />		grpOriginal
<OriginalCityAbbreviation />		grpOriginal
<OriginalCountryCode />		grpOriginal
<OriginalPlus4 />		grpOriginal
<OriginalPostalCode />		grpOriginal
<OriginalPrivateMailBox />		grpOriginal
<OriginalResults />		grpOriginal
<OriginalState />		grpOriginal
<OriginalStateName />		grpOriginal
<OriginalSuite />		grpOriginal
<OriginalUrbanizationName />		grpOriginal

Column Name	Default	Group Name
<Plus4 />		ColumnOnly
<PostalCode />	Default	
<PrivateMailBox />		ColumnOnly
<RecordID />	Default	
<Results />	Default	
<StandardizedAddressLine1 />		grpStandardized
<StandardizedAddressLine2 />		grpStandardized
<StandardizedCarrierRoute />		grpStandardized
<StandardizedCity />		grpStandardized
<StandardizedCityAbbreviation />		grpStandardized
<StandardizedCountryCode />		grpStandardized
<StandardizedDeliveryPointCheckDigit />		grpStandardized
<StandardizedDeliveryPointCode />		grpStandardized
<StandardizedPlus4 />		grpStandardized
<StandardizedPostalCode />		grpStandardized
<StandardizedPrivateMailBox />		grpStandardized
<StandardizedResults />		grpStandardized
<StandardizedState />		grpStandardized
<StandardizedStateName />		grpStandardized
<StandardizedSuite />		grpStandardized
<StandardizedUrbanizationName />		grpStandardized
<State />	Default	
<StateName />	Default	
<Suite />		ColumnOnly
<UrbanizationName />	Default	