Reference Guide

Address Object

32/64 BIT

Multiplatform

MELISSA DATA®

Address Object

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/addressobject.htm.

Document Code: ADORFG
Revision Number: 240409.276
Last Update: April 9, 2024

Dear Developer,

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

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

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

I look forward to hearing from you.

Best Wishes,

Raymond F. Melissa
President/CEO

Table of Contents

Chapter 1
Welcome to Address Object

These customizable developer tools have been designed to help you efficiently manage your contact data for superior performance and profitability. This is the crucial point that turns collected data into usable information. Depending on the objects you have purchased from this Suite, you will be able to verify and standardize addresses, validate phone numbers, update area codes, parse and genderize names, validate email addresses, and more.

1.1: About Address Object

Clean up contact data of bad, incomplete information before it invades your database and creates a negative impact on your data-driven initiatives. You'll reduce undeliverables, increase communication efforts, and save money on all your direct marketing and CRM campaigns.

Address Object is available for Microsoft Windows, Linux IA-32, Solaris SPARC, HP-UX, and AIX. It can be used with virtually any programming language that supports object-oriented programming, such as Visual Basic, C++, and C.

Address Object's functionality is divided into four interfaces, AddressCheck, Parse, StreetData, and Data.

- **AddressCheck** can verify that an address is a properly formatted address that matches a true and valid point of delivery using the DPV® database. It also matches the address to the LACS$^{Link®}$ file to determine if the address has been

converted (usually from a rural route to a standard city-street address) and automatically updates the address.

The CASS Certified™ address checking logic includes DPV, LACS^{Link}, and Suite$^{Link®}$ processing mandatory to qualify your mailings for postal discounts by appending the ZIP + 4® codes to only validated addresses. Address Object also generates Form 3553 required by the Postal Service™ to claim the discounts.

The AddressCheck Interface also enhances your data by providing additional information, such as Congressional District, County name, FIPS code, time zone, and others.

- The **StreetData** Interface can match a street name or just a partial name against a ZIP Code™ and return the known, valid address ranges that match that pattern.

  Used in conjunction with the AddressCheck Interface, StreetData can be used to match incorrect or misspelled addresses to a valid address range. For example, if “123 Main” matches both “123 Main St” and “123 Main Ave” in the same ZIP Code, the AddressCheck Interface would return a multiple match error. The StreetData Interface can return all records that match that pattern in the same ZIP Code, allowing you to select an address record that would otherwise result in an undeliverable mail piece.

- The **ZipData** Interface performs several functions, matching geographic data to ZIP Code and city information.

  First, the **FindZip** function returns official and unofficial city names that match a ZIP Code, as well as postal facility and geographic data for those cities, including automation (or carrier route) information, county name, and FIPS code, postal facility type, official “last line” indicator and latitude/longitude for each match. The **FindZipInCity** functions can return the same information for every ZIP Code that matches that input city and state combination. The **FindCityInState** functions can determine if the input full or partial city name matches a valid city name within the state. This can be useful to help correct misspelled city names.

- The **Parse** Interface breaks down a street address into its component parts, such as street number, directional (S, NW, and so on), street name, and suffix (ST, RD, BLVD).

  If the first pass does not yield the expected results because the street address follows a non-standard format, the **ParseNext** function will offer an alternate parsing.

  The AddressCheck Interface also parses the submitted data, but the Parse Interface can be used without initializing the data files required for AddressCheck, making Parse faster to use when full address checking is not necessary.

This interface can also parse Last Line data (city, state, and ZIP Code) into the separate parts. This is handy if your address data comes to you as lines of plain text instead of discrete fields.

Best Practice #1: Add-ons to Address Object

- **Canadian Add-on** — Increase the accuracy of your Canadian addresses. Use this API with your custom PC & Web applications to validate, correct, and standardize Canadian addresses.
- **RBDI** (Residential Business Delivery Indicator) — Most shipping carriers charge a higher price for residential deliveries. RBDI helps ensure that you select the most cost-effective function for shipping parcels by identifying residential addresses... therefore maximizing your savings potential.
- **AddressPlus** — AddressPlus goes beyond the Post Office's Suite^{Link} product, appending missing secondary address information to millions of business and residential addresses.

1.2: Entering Your Address Object License

The license string is a software key that unlocks the full functionality of the component. Without the license string, Address Object will only function in demo mode.

You set the license string using an environment variable called MD_LICENSE. If you are just trying out Address Object and have a demo license, you can use the environment variable MD_LICENSE_DEMO for this purpose. This avoids conficts or confusion if you already have active subscriptions to other Melissa Data object products.

When using the demo license, Address Object will only code addresses located in Nevada.

In earlier versions of Address Object, you would set this value with a call to the **SetLicenseString** function. Using an environment variable makes it much easier to update the license string without having to edit and re-compile the application.

It used to be necessary, even when employing an environment variable, to call the **SetLicenseString** function without passing the license string value. This is no longer true. Address Object will still recognize the **SetLicenseString** function, but you should eventually remove any reference to it from your code.

Windows

Windows users can set environment variables by doing the following:

1. Select **Start > Settings**, and then click **Control Panel**.
2. Double-click **System**, and then click the **Advanced** tab.
3. Click **Environment Variables**, and then select either *System Variables* or *Variables for the user X*.

4 Click **New**.

5 Enter “MD_LICENSE” in the *Variable Name* box.

6 Enter the license string in the *Variable Value* box, and then click **OK**.

Please remember that these settings take effect only upon start of the program. It may be necessary to quit and restart the application to incorporate the changes.

Linux/Solaris/HP-UX/AIX

Unix-based OS users can simply set the license string via the following (use the actual license string, instead):

```
export MD_LICENSE=A1B2C3D4E5
```

If this setting is placed in the .profile, remember to restart the shell.

Address Object also used to employ its own environment variable, MDADDR_LICENSE. The MD_LICENSE variable is shared across the entire Melissa Data product line of programming tools. Address Object will still use the old license variable for the time being, but you should transition to using MD_LICENSE as soon as possible.

1.3: Using Results Codes: Coding for the present and the future

Over a year ago, Melissa Data introduced a new concept know as results 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 results 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 results 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 Results codes, but you won’t use them all.

The first step to understanding how to use Results 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 deliver using USPS or don't deliver at all, but it would be important if you deliver using a third party carrier, like UPS.

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

The ultimate goal of using Results 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” records 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 into 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"))
```

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

Since the inception of Results 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 Results codes is to actually try and use them. See what Results codes are produced by different types of addresses, and how your code handles them. For an up-to-date online reference of all Results codes available and examples to produce each code, visit here:

http://www.melissadata.com/tech/ResultCodes.asp.

CHAPTER 2
AddressCheck Interface

The **AddressCheck** Interface verifies and standardizes your address data using the most current data from the U. S. Postal Service®. The programming logic used by AddressCheck is CASS Certified™. This stringent certification ensures the quality of the data that is passed through the AddressCheck Interface and must be renewed every year.

Use the AddressCheck Interface of Address Object to verify addresses in real time, either as part of your in-house data entry system or when gathering customer data via your website. Standardizing street names and abbreviations will also help eliminate duplicate entries.

If CASS™ processing is enabled, the object will also generate CASS Form 3553, which certifies that your data has been checked against Post Office™ data using CASS Certified software, which enables you to receive discounted postal rates.

2.1: Address Handling

A key concept for Address Object is understanding how the service handles address data. Address Object can accept two lines of street address information via the **SetAddress** and **SetAddress2** functions.

The value passed to the **SetAddress** function will typically contain the primary street address (Street number, street name, plus any directionals and street suffixes). It may or may not contain a secondary address, such as a suite number, address number, unit number, or a private mailbox located at a Commercial Mail Receiving Agency (CMRA).

The value passed to the **SetAddress2** function will often contain the secondary address information, if it is not submitted via the **SetAddress** function or the **SetSuite**

function. The **SetAddress2** function 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 secondary address information is submitted via the **SetAddress2** function, Address Object will append this information to the value passed to the **SetAddress** function before processing the record.

Example #1

If the following were submitted:

Address Line 1: 1234 Main Street
Address Line 2: Suite #101

This is the address that would actually be verified:

Address Line 1: 1234 Main Street Suite #101
Address Line 2: <empty>

If an additional primary address was sent to the **SetAddress2** function, such as a P.O. Box or a second street address, and the address sent to the **SetAddress** function cannot be verified, then Address Object will attempt to verify the second address line.

Example #2

If the following were submitted:

Address Line 1: 1234 Main Street
Address Line 2: P.O. Box 101

Assuming that the first address didn't contain a verifiable address, this is when Address Object would consider the contents of the second address line.

If the second address line contains a verifiable address, then the values submitted via the **SetAddress2** function will be used. In that case, Address Object will swap the values sent via the **SetAddress** and **SetAddress2** function and the object will return the standardized contents of these fields in this order:

The **GetAddress** function returns "P.O.Box 101"
The **GetAddress2** function returns "1234 Main St"

If the contents of neither field can be successfully verified, then Address Object will flag the error and the values submitted to the **SetAddress** and **SetAddress2** functions will be returned in their **original** order:

The **GetAddress** function returns "1234 Main Street"
The **GetAddress2** function returns "P.O.Box 101"

Secondary Addresses

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

The secondary address can be passed to Address Object at the end the first address line, as the second address line, or via the **SetSuite** function.

The National Postal database identifies certain primary addresses as highrises, business parks, and apartment buildings. Therefore, Address Object 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 Object 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 **GetPrivateMailbox** function rather than the **GetSuite** function.

Be aware that Address Object 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 Object will identify the “#101” portion of the second line as a PMB number and return this number in the Private Mailbox field.

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 an address, Address Object requires information on the city, the state or province, and a ZIP™ or Postal code. With the city and the state/province, the Address Object will determine the correct ZIP/Postal code and use this to verify the address.

Conversely, if supplied with a correct ZIP/Postal code, Address Object 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 Object will use the city and state/province to look up the ZIP/Postal code.

DeliveryPlus — Non-USPS Addresses

Infrequently, an address that physically exists will not be serviced by the U.S. Postal Service®. These addresses may still be serviced by delivery carriers such as DHL, UPS, or FedEx. With DeliveryPlus, Address Object will verify such an address, but the **GetResults** function will return a results code of “AS03” instead of “AS01.”

Address Object will not return the same level of information for a non-USPS address that it would for an address serviced by the Post Office. See the **VerifyAddress** function on page 62 to see which functions will or will not return a value for a non-USPS address.

2.2: Using the AddressCheck Interface

1 Create an instance of the Address Object’s AddressCheck Interface.

```
Set addPtr = new instance of AddressCheck
```

The first set of steps is to initialize the instance of the AddressCheck Interface. For more information on the functions used in these steps, see pages 20 through 34.

2 Call the **SetPathToUSFiles** function and, if necessary, the **SetPathToCanadaFiles** function, with paths pointing to the following data files:

SetPathToUSFiles — mdAddr.dat, mdAddr.nat, mdAddr.lic, and mdAddr.str

SetPathToCanadaFiles — mdCanada3.db.

```
// set required paths for CASS Processing
```

```
CALL SetPathToUSFiles WITH PathToUSDataFiles
CALL SetPathToDPVDataFiles WITH PathToDPVFiles
CALL SetPathToLACSLinkDataFiles WITH PathToLACSLinkFiles
CALL SetPathToSuiteLinkDataFiles WITH PathToSuiteLinkFiles
```

3 Call the **InitializeDataFiles** function. If it returns anything other than 0, then initialization has failed. Check the return value of the **GetInitializeErrorString** function to find out why this happened.

```
CALL InitializeDataFiles RETURNING result
If result <> 0 THEN
   Call GetInitializeErrorString RETURN ErrorMessage
   Display ErrorMessage
Else
   Process Addresses
End If
```

4 At this point, select from several optional settings available for the AddressCheck Interface. For more information on these functions, see pages 35 through 38.

```
CALL UseUSPSPreferredCityNames WITH TRUE
CALL SetStandardizationType WITH LongFormat
```

5 To use Address Object for CASS processing, you must explicitly call the **SetCASSEnable** function and supply the list owner name and address. For more information on these functions, see page 41 through 44.

```
CALL SetCASSEnable WITH TRUE
CALL SetPS3553_B1_ProcessorName WITH ProcessorName
CALL SetPS3553_B4_ListName WITH ListName
CALL SetPS3553_D3_Name WITH ListOwnerName
CALL SetPS3553_D3_Address WITH strListOwnerAddr
CALL SetPS3553_D3_City WITH ListOwnerCity
CALL SetPS3553_D3_State WITH ListOwnerState
CALL SetPS3553_D3_ZIP WITH ListOwnerZip
```

6 Call the required functions with the address data to be coded. The **ClearProperties** function ensures that there is no data left over from a previous call. For more information on these functions, see pages 47 through 54.

```
CALL ClearProperties
CALL SetCompany WITH CompanyName
CALL SetAddress WITH AddressLine1
CALL SetAddress2 WITH AddressLine2
CALL SetSuite WITH SuiteNumber
CALL SetCity WITH City
CALL SetState WITH State
CALL SetZip WITH Zip5
```

A suite number can be passed at the end of the values passed to the **SetAddress** function, or as the parameter of either the **SetAddress2** or the **SetSuite** function.

If the value passed to the **SetAddress** function cannot be verified, Address Object will attempt to verify the value passed via the **SetAddress2** function. See *Address Handling* on page 7 for more information

If you call the **SetZip** function, the **SetCity** and SetState functions are optional. Likewise, if City and State are populated, **SetZip** is optional. If possible, it is the best practice to pass all three values if possible, because Address Object will use the values to validate each other.

Another option, if your city, state, and ZIP information comes as a single string, is to call the **SetLastLine** function in place of **SetCity**, SetState, and **SetZip**.

```
CALL SetPlus4 WITH Plus4
CALL SetUrbanization WITH Urbanization
CALL SetCountryCode WITH CountryCode
```

7 Call the **VerifyAddress** function and read the resulting data.

```
CALL VerifyAddress
CALL GetResults RETURNING ResultCodes
IF ResultCodes CONTAIN "AS01" THEN
   CALL GetCompany RETURNING CompanyName
   CALL GetAddress RETURNING AddressLine1
   CALL GetSuite RETURNING SuiteNumber
   CALL GetCity RETURNING CityName
   CALL GetState RETURNING State
   CALL GetPrivateMailbox RETURNING PrivateMailbox
   CALL GetZip RETURNING Zip5
   CALL GetPlus4 RETURNING Plus4
   CALL GetCarrierRoute RETURNING CarrierRoute
   //
   // See page 65 for a full list of all functions returning
   // data.
   //
   Write Fields Back To Database
ELSE
   PROCESS ResultCodes FOR Error Messages
END IF
```

8 If the submitted address could not be verified, the **FindSuggestion** function can assist in finding the most likely alternative.

```
CALL FindSuggestion RETURNING SuggestionFound
```

If the function returns 0, no suggestion was found.

If the function returns an integer value of 1, then a suggestion was found. Repeat the function calls in the previous step to retrieve the suggested address data.

Check for further suggestions using the **FindSuggestionNext** function.

```
CALL FindSuggestionNext RETURNING SuggestionFound
```

If the function returns 0, no suggestion was found.

If the function returns an integer value of 1, then a suggestion was found. Repeat the function calls in the previous step to retrieve the suggested address data.

9 Finally, generate your CASS Form 3553 to take to the Post Office when you use this list for a mailing.

```
CALL SaveFormPS3553 WITH CASSFormPath
```

If using the Canadian add-on, use the Summary of Addresses (SOA) functions, instead.

```
CALL SaveFormSOA WITH SOAFormPath
```

Address Object returns values to many more additional functions, which you can use to further enhance your address data, if needed.

2.3: AddressCheck Interface Functions

2.3.1: Initialize AddressCheck

The following functions set the paths to the necessary data files, the license, and initialize the AddressCheck Interface:

2.3.2: Set AddressCheck Options

The following functions enable or disable the optional functionality of the AddressCheck Interface:

2.3.3: Configure the CASS Form 3553

U.S. Only — The following functions are used to populate user-configurable values required for CASS Form 3553:

2.3.4: Configure the SOA Form

Canada Only — The following functions are used to populate and retrieve the Canada Post Summary of Addresses if you are using the Canadian add-on for Address Object:

2.3.5: Supply the Address Information to Be Verified

The following functions supply the address information to be verified by a call to the VerifyAddress function. After the function call, the return values of these functions will be populated with the verified and standardized address data:

2.3.6: Other Input Functions

The following functions are not always used for address verification. The SetLastName function is used if you are using Address Object with the AddressPlus option. The parsed address functions can be used in place of the SetAddress, SetAddress2, and SetSuite functions if your address information is already parsed:

2.3.7: Verify the Address Data

The VerifyAddress function standardizes and verifies the submitted address information.

2.3.8: Retrieve Status and Return Codes

The following functions return information about the level and types of data matches found by a call to the VerifyAddress function:

2.3.9: Retrieve the Standardized and Verified Address Data

The following functions retrieve the address information that has been verified and standardized by a call to the VerifyAddress function:

2.3.10: Retrieve Parsed Street Address

In addition to verifying and standardizing an address, the AddressCheck Interface will also parse the street address and return the parts via the following functions:

2.3.11: Retrieve Demographic and Other Information

The following functions return additional information linked to the submitted address:

2.3.12: Retrieve Information for Mailing and Sorting

The following functions return information used for presorting lists and printing address labels:

2.3.13: Suggestions

The following functions return possible correct addresses if the VerifyAddress function was unable to verify or correct the submitted address:

2.3.14: Retrieve CASS Form 3553

U.S. Only — The following functions retrieve CASS Form 3553 and return the information used to populate the form. Address Object only generates a CASS Form 3553 for U.S. addresses. This information is already included on the generated CASS Form, so it is returned here only for information purposes:

2.3.15: Retrieve Summary of Addresses

Canada Only — The following functions retrieve the Canada Post Summary of Addresses (SOA) and return data used to populate the form. These functions only apply when using the Canadian add-on for verifying Canadian addresses:

2.3.16: Melissa Address Key

The Melissa Address Key (MAK) is a new output available only from Melissa data.

2.3.1: Initialize AddressCheck

The following functions set the paths to the necessary data files, the license, and initialize the AddressCheck Interface:

SetPathToCanadaFiles

Canada Only — This function must be set to the path to the following files used for Canadian address validation:

```
mdCanada3.db
```

For information about purchasing the Canadian add-on, call 1-800-MELISSA.

Remarks

You do not need to call this function if you are only coding U.S. addresses.

The Canadian data files expire every 30 days, in accordance with Canada Post regulations. If you do not receive an e-mail notification instructing you to download the most current Canadian data files, please contact your sales representative.

Syntax

```
object->SetPathToCanadaFiles(StringValue);
```

C

```
mdAddrSetPathToCanadaFiles(object, char*);
```

COM

```
object.PathToCanadaFiles = StringValue
```

SetPathToDPVDataFiles

U.S. Only — Required for CASS Processing. The string passed to this function must contain the path to the following files, which are required for DPV:

dph.hsa	dph.hsx	dph.hsv
dph.hsf	dph.hsc	dph.dte
lcd		

Remarks

If this path is not set, DPV matching of your addresses will not occur. DPV is available for U.S. addresses only.

Note: The date of the DPV files must match the date of the ZIP + 4 file in the path passed to **SetPathToUSFiles** function.

Syntax

```
object->SetPathToDPVDataFiles(StringValue);
```

C

```
mdAddrSetPathToDPVDataFiles(object, char*);
```

COM

```
object.PathToDPVDataFiles = StringValue
```

SetPathToLACSLinkDataFiles

U.S. Only — Required for CASS Processing. The string passed to this function must contain the path to the following file used by the LACS^{Link} features:

```
mdLACS.dat
```

Remarks

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

The LACS^{Link} service matches the old address with the updated address and corrects it as part of the Address Check process.

If this path is not set, LACS^{Link} correction of your addresses will not occur. LACS^{Link} is available for U.S. addresses only.

Syntax

```
object->SetPathToLACSLinkDataFilesFiles(StringValue);
```

C

```
mdAddrSetPathToLACSLinkDataFilesFiles(object, char*);
```

COM

```
object.PathToLACSLinkDataFiles = StringValue
```

SetPathToRBDIFiles

For information about purchasing the RBDI add-on for Address Object, call 1-800-MELISSA.

U.S. Only — The string passed to this function must contain the path to the following data file, which is required for the **GetRBDI** function:

```
mdRBDI.dat
```

Remarks

If this path is not set, the **VerifyAddress** function will not populate the **GetRBDI** function. RBDI is available for U.S. addresses only.

The RBDI files must have been included with the same update as the ZIP + 4 files in the path passed to the **SetPathToUSFiles** function.

Syntax

```
Object->SetPathToRBDIFiles(StringValue);
```

C

```
mdAddrSetPathToRBDIFiles(object, char*);
```

COM

```
Object.PathToRBDIFiles = StringValue
```

SetPathToSuiteFinderDataFiles

For information about purchasing the AddressPlus add-on for Address Object, call 1-800-MELISSA.

The string passed to this function must contain the path to following data files, which is required to use the AddressPlus add-on:

```
mdSuiteFinder.db
```

Requires
AddressPlus add-on for Address Object

Remarks

If this path is not set, the **VerifyAddress** function will not return an appended residential suite number, if available. AddressPlus is available for U.S. addresses only.

Syntax

```
Object->SetPathToSuiteFinderDataFiles(StringValue);
```

C

```
mdAddrSetPathToSuiteFinderDataFiles(object, char*);
```

COM

```
Object.PathToSuiteFinderDataFiles = StringValue
```

SetPathToSuiteLinkDataFiles

U.S. Only — Required for CASS Processing. The string passed to his function must contain the path to the following file, which isrequired for Suite^{Link}:

```
mdSteLink.dat
```

Remarks

If this path is not set, Suite^{Link} matching or updating of your addresses will not occur. Suite^{Link} is required for CASS Processing as of CASS Cycle M, 2009. Suite^{Link} is available for U.S. addresses only.

Note: The date of the Suite^{Link} files must match the date of the ZIP + 4 file in the path passed to the **SetPathToUSFiles** function.

Syntax

```
Object->SetPathToSuiteLinkDataFiles(StringValue);
```

C

```
mdAddrSetPathToSuiteLinkDataFiles(object, char*);
```

COM

```
Object.PathToSuiteLinkDataFiles = StringValue
```

SetPathToUSFiles

U.S. Only — Required for CASS Processing. The string passed to this function must contain the path to the data files required for U.S. address verification:

mdAddr.dat	mdAddr.nat	mdAddr.str
mdAddr.lic		

Remarks

You do not need to call this function if you are only coding Canadian addresses with the Canadian add-on.

Syntax

```
object->SetPathToUSFiles(StringValue);
```

C

```
mdAddrSetPathToUSFiles(object, char*);
```

COM

```
object.PathToUSFiles = StringValue
```

SetLicenseString

This function sets the license string required to enable Address Object's complete functionality.

Remarks

The License String is a software key that unlocks the full functionality of the component. Without the License String, the object will only function in DEMO mode (Nevada only).

The license string is normally set using an environment variable, either MD_LICENSE or MD_LICENSE_DEMO. Calling **SetLicenseString** is an alternative method for setting the license string, but applications developed for a production environment should only use the environment variable.

When using an environment variable, it is not necessary to call the **SetLicenseString** function.

For more information on setting the environment variable, see page 3 of this guide.

Input Parameters

The SetLicenseString function has one parameter:

LicenseString	A string value representing the software license key.

Return Value

The **SetLicenseString** function returns a Boolean value of 0 (FALSE) or 1 (TRUE). The **SetLicenseString** function will return a FALSE Boolean value if the License String provided is incorrect.

Syntax

```
BooleanValue = object->SetLicenseString(LicenseString);
```

C

```
IntegerValue = mdAddrSetLicenseString(object, LicenseString);
```

COM

```
BooleanValue = object.SetLicenseString(LicenseString)
```

SetCASSEnable

U.S. Only — Enables or disables full CASS Certified address verification. This option is disabled by default.

Remarks

When the value passed to this function is TRUE, Address Object performs full CASS Certified address verification, including DPV, when the **VerifyAddress** function is called.

This function must be called before the **InitializeDataFiles** function is called.

If the Boolean parameter passed to this function is TRUE:

- Linking to DPV, LACS^{Link} and Suite^{Link} data files is required to initialize.
- If an address cannot be DPV verified, the address will not be ZIP + 4 coded by the **VerifyAddress** function.
- A CASS Form 3553 can be generated.

If the parameter passed to this function is FALSE:

- Linking to DPV, LACS^{Link} and Suite^{Link} data files is optional.
- The **GetPlus4** function will return a value for all Addresses verified to ZIP + 4 level or higher.
- A CASS Form 3553 cannot be generated.
- By default, CASS processing is disabled and must be explicitly enabled by passing TRUE to this function.

Syntax

```
object->SetCASSEnable(int);
```

C

```
mdSetCASSEnable(object, int);
```

COM

```
object.CASSEnable = Boolean
```

InitializeDataFiles

Opens the required data files and prepares the address checking logic for use.

You MUST call the **SetLicenseString** function before the **InitializeDataFiles** function if you do not intend to run in DEMO mode.

Remarks

If the **InitializeDataFiles** function returns a code other than zero, you can call the **GetInitializeErrorString** function to display a string describing the error. The Canadian data files expire every 30 days, in accordance with Canada Post regulations. If you do not receive an e-mail notification instructing you to download the most current Canadian data files, please contact your sales representative.

Required Functions

You must successfully call either the **SetPathToUSFiles**, the **SetPathToCanadaFiles**, or both functions in order to successfully call the **InitializeDataFiles** function.

Return Value

The **InitializeDataFiles** function returns one of the following:

Code	Description
0	No error - initialization was successful.
1	Could not open the `mdAddr.dat` file.
2	Could not open the `mdAddr.nat` or `mdAddr.str` file.
4	The internal database date of the `mdAddr.dat` and `mdAddr.nat` files do not match.
5	Not all the memory buffers could be initialized.
6	Unknown error.
7	Error in Canada File.
8	Expired Database.
9	Canadian Database expired.
10	DPV data files error.
11	RBDI data files error.
12	RBDI and US Data File dates do not match.
13	LACS^{Link} data files error.
14	Suite^{Link} data files error.
15	AddressPlus data files error.

Syntax

```
StringValue = object->InitializeDataFiles();
```

C

```
StringValue = mdAddrInitializeDataFiles(object);
```

COM

```
StringValue = object.InitializeDataFiles()
```

GetInitializeErrorString

This function returns a descriptive string to describe an error from the **InitializeDataFiles** function.

Remarks

The **GetInitializeErrorString** function returns a string describing the error that occurred when the **InitializeDataFiles** function failed. This is useful for outputting a quick message to the user.

Syntax

```
StringValue = object->GetInitializeErrorString();
```

C

```
StringValue = mdAddrGetInitializeErrorString(object);
```

COM

```
StringValue = object.GetInitializeErrorString()
```

GetBuildNumber

The **GetBuildNumber** function returns a string value containing the current development release build number of the Address Object.

Remarks

This is usually a three or four-character string. If you are running the Demo version of Address Object, the word DEMO will be appended to the build number.

Return Value

The **GetBuildNumber** function returns a string value containing the current development release build number of the Address Object.

Syntax

```
StringValue = object->GetBuildNumber();
```

C

```
StringValue = mdAddrGetBuildNumber(object);
```

COM

```
StringValue = object.GetBuildNumber()
```

GetCanadianDatabaseDate

Canada Only — The **GetCanadianDatabaseDate** function returns a value representing the date of your Canadian data files. This date allows you to confirm that the data files you are using are the latest available.

Remarks

If the **GetCanadianDatabaseDate** function is called before the **InitializeDataFiles** function is called, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

Return Value

The **GetCanadianDatabaseDate** function returns a value representing the date of the Canadian data files. The standard object returns a string value while the COM object returns a date value.

Syntax

```
StringValue = object->GetCanadianDatabaseDate();
```

C

```
StringValue = mdAddrGetCanadianDatabaseDate(object);
```

COM

```
DateTime = object.GetCanadianDatabaseDate()
```

GetCanadianExpirationDate

Canada Only — The **GetCanadianExpirationDate** function returns a date value representing the date your Canadian data files expire. This date allows you to confirm that the data files you are using are the latest available.

Remarks

If the **GetCanadianExpirationDate** function is called before the **InitializeDataFiles** function is called, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

Return Value

The **GetCanadianExpirationDate** function returns a value representing the expiration date of the Canadian data files. The standard object returns a string value while the COM object returns a date value.

Syntax

```
StringValue = object->GetCanadianExpirationDate();
```

C

```
StringValue = mdAddrGetCanadianExpirationDate(object);
```

COM

```
DateTime = object.GetCanadianExpirationDate()
```

GetDatabaseDate

The **GetDatabaseDate** function returns the date of your U.S. address data files. This date allows you to confirm that the data files you are using are the latest available.

Remarks

If the **GetDatabaseDate** function is called before the **InitializeDataFiles** function is called, and the data files are not in the same directory as Address Object, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

This function is only used with the Standard DLL. When using the COM object, call the **GetUSDatabaseDate** method.

Return Value

The **GetDatabaseDate** function returns a string value that represents the date of your address data files.

Syntax

```
StringValue = object->GetDatabaseDate();
```

C

```
StringValue = mdAddrGetDatabaseDate(object);
```

GetExpirationDate

U.S. Only — The **GetExpirationDate** function returns a date value representing the date when the current U.S. data files expire. This date enables you to confirm that the data files you are using are the latest available.

Remarks

If the **GetExpirationDate** function is called before the **InitializeDataFiles** function is called, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

This function is only used with the Standard DLL. When using the COM object, call **GetUSExpirationDate**.

Return Value

The **GetExpirationDate** function returns a string value representing the expiration date of the U.S. data files.

Syntax

```
StringValue = object->GetExpirationDate();
```

C

```
StringValue = mdAddrGetExpirationDate(object);
```

GetRBDIDatabaseDate

U.S. Only — The **GetRBDIDatabaseDate** function returns a date value representing the date of your RBDI data files. This date allows you to confirm that the data files you are using are the latest available.

Requires

RBDI add-on for Address Object

For more information about purchasing the RBDI add-on for Address Object, call 1-800-MELISSA.

Remarks

If the **GetRBDIDatabaseDate** function is called before the **InitializeDataFiles** function is called, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

Return Value

The **GetRBDIDatabaseDate** function returns a value representing the date of the RBDI data files. The standard object returns a string value while the COM object returns a date value.

Syntax

```
StringValue = object->GetRBDIDatabaseDate();
```

C

```
StringValue = mdAddrGetRBDIDatabaseDate(object);
```

COM

```
date = object.GetRBDIDatabaseDate()
```

GetUSDatabaseDate

U.S. Only — The **GetUSDatabaseDate** function returns a date value representing the date of your U.S. data files. This date allows you to confirm that the data files you are using are the latest available.

Remarks

If the **GetUSDatabaseDate** function is called before the **InitializeDataFiles** function is called, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

This function is only used with the COM Object. When using the Standard DLL, call the **GetDatabaseDate** function.

Return Value

The **GetUSDatabaseDate** function returns a date value representing the date of the U.S. data files.

Syntax

```
DateTime = object.GetUSDatabaseDate()
```

GetUSExpirationDate

U.S. Only — The **GetUSExpirationDate** function returns a date value representing the date when the current U.S. data files expire. This date enables you to confirm that the data files you are using are the latest available.

Remarks

If the **GetUSExpirationDate** function is called before the **InitializeDataFiles** function is called, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

This function is only used with the COM Object. When using the Standard DLL, call the **GetExpirationDate** function.

Return Value

The **GetUSExpirationDate** function returns a date value representing the expiration date of the U.S. data files.

Syntax

```
DateTime = object.GetUSExpirationDate()
```

GetLicenseExpirationDate

This function returns a date value corresponding to the date when the current license string expires.

Remarks

License strings issued by Melissa Data are valid a certain period of time. This function returns the date after which the current license string is no longer valid.

The COM object returns a date value, while the standard object returns a string value.

Syntax

```
StringValue = object->GetLicenseExpirationDate();
```

C

```
StringValue = mdAddrGetLicenseExpirationDate(object);
```

COM

```
DateTime = object.GetLicenseExpirationDate
```

2.3.2: Set AddressCheck Options

The following functions enable or disable the optional functionality of the AddressCheck Interface:

DisableAddressSwapping

COM Object only. This function does not appear in the cross-platform object, which will always perform address swapping.

The **DisableAddressSwapping** function disables swapping of the parameters passed to the **SetAddress** and **SetAddress2** functions when the value passed to the **SetAddress** function cannot be coded.

By default, the COM object performs address swapping.

Remarks

When the **VerifyAddress** function is called, Address Object will attempt to code the address passed to the **SetAddress** function with the parameters passed to the **SetCity**, **SetState**, and **SetZip** functions. If this address is not successfully coded, Address Object will attempt to code using the string parameter passed to the **SetAddress2** function, if any.

If the data is successfully coded using the **SetAddress2** function's parameter, Address Object will swap the parameters passed to the **SetAddress** and **SetAddress2** function, so that the primary address for this record is a properly coded address.

If the **DisableAddressSwapping** function is set to TRUE, Address Object will not swap the two parameters and return an error code of 2. This function defaults to FALSE unless explicitly set to TRUE.

Address swapping can be re-enabled by setting this function to FALSE.

Input Parameters

TRUE or FALSE

Return Value

None

Syntax

COM Only

```
object.DisableAddressSwapping(TRUE or FALSE)
```

EnableEarlyWarningSystem

COM Object Only. Enables checking for new addresses using the Early Warning System (EWS) data. This feature is automatically on for the cross-platform object. To use, simply place the most up-to-date ews.txt file in the same directory selected by the **SetPathToUSFiles** function.

Remarks

EWS (Early Warning System) data is composed of new addresses (for example, new housing developments) that are scheduled for inclusion in the USPS national database. The USPS updates this data every week to reflect new construction and home ownership. When verifying an address, Address Object consults the EWS file as well. Having an updated EWS file prevents Address Object from miscoding new address information with data in the older USPS national database file. For example, an input of 44 Legacy Drive would be changed to 44 Legacy Street if Legacy Drive does not yet exist in the USPS national database. If Legacy Drive is in the EWS file, however, Address Object would recognize it as a new street and not code the record. The next database update should contain the new information and enable Address Object to properly code the record.

Be sure to frequently download the latest EWS file in order to have the most current address records. However, using the EWS file is optional. Current EWS files are available every Thursday. The EWS file can be downloaded from ftp://ftp.melissadata.com/Updates/ews.txt.

If this function is not called with a TRUE value, the EWS properties of the COM Object (EWSCount and EWSFlag) will not return a value.

Input Parameters

Boolean value, TRUE or FALSE

Return Value

None

Syntax

COM Only

```
object.EnableEarlyWarningSystem(TRUE)
```

UseUSPSPreferredCityNames

U.S. Only — When the Boolean value passed to this function is TRUE, the **VerifyAddress** function will always return the preferred city name for the submitted address, regardless of whether or not an approved vanity name was sent to the function.

Remarks

For every city, there is an official name that is preferred by the U.S. Postal Service. There may be one or more unofficial or "vanity" names in use. Normally, Address Object allows you to verify addresses using known vanity names. If the value passed to this function is TRUE, Address Object will substitute the preferred city name for any vanity name when it verifies an address.

Syntax

```
object->SetUseUSPSPreferredCityNames(bool);
```

C

```
mdAddrSetUseUSPSPreferredCityNames(object, int);
```

COM

```
object.UseUSPSPreferredCityNames = Boolean
```

SetDiacritics

Canada Only — This function enables or disables the use of French-language characters that might be found in addresses located in Quebec.

Remarks

The **SetDiacritics** function accepts a single value that sets how Address Object will handle diacritic characters in French words for addresses located in Quebec. For programming languages that handle enumerated values, this parameter uses the data type DiacriticsMode.

Enumerated Value	Integer Value	
Auto	0	If the input value contains diacritic characters, Address Object will use diacritics in the output. If it does not, it will not use diacritics.

Enumerated Value	Integer Value	
On	1	The return values will always have diacritics and all addresses will be returned in the French format.
Off	2	The return values will never have diacritics and all addresses will be returned in the English format.

Languages that do not use enumerations will use the equivalent integer value.

Syntax

```
object->SetDiacritics(DiacriticsMode);
```

C

```
mdAddrSetDiacritics(object, int);
```

COM

```
object.SetDiacritics = DiacriticsMode
```

SetStandardizationType

This function controls how Address Object handles the abbreviations of suffixes and directionals when standardizing a street address.

Remarks

The **SetStandardizationType** function accepts an enumerated value of the type StandardizeMode. There are three settings:

ShortFormat (0)

The street address is returned with directionals and suffixes shortened according to postal standards. This is the default setting.

100 West Main Street East

...is returned as:

100 W Main ST E

LongFormat (1)
The street address is returned with directionals and suffixes spelled out rather than abbreviated.

100 W Main ST E

...is returned as:

100 West Main Street East

AutoFormat (2)
The street address is returned with the abbreviation or non-abbreviation of the directionals and suffixes preserved. If any of these components need to be corrected and replaced, the replacement will preserve the original standardization.

100 W Main Ave East

...is returned as:

100 W Main ST East

These settings also affect the output of the parsed street address functions. If the selected setting is LongFormat, and the address is "100 W Main ST E," the **GetParsedPostDirection** function will return "East."

Syntax

```
object->SetStandardizationType(StandardizeMode);
```

C

```
mdAddrSetStandardizationType(object, int);
```

COM

```
object.StandardizationType = StandardizeMode
```

SetSuiteParseMode

This function controls how Address Object handles whether or not the suite information will be parsed out into its own field or appended to the end of the Address1 line after VerifyAddress().

Remarks

The **SetSuiteParseMode** function accepts one of two set values of the type SuiteParseMode.These values are:

ParseSuite (Default)
The suite will be parsed out into the suite field after VerifyAddress(). This is the default setting.

CombineSuite
The suite will be appended to the end of the Address1 line after VerifyAddress().

Syntax

```
object->SetSuiteParseMode(SuiteParseMode);
```

C

```
mdAddrSetSuiteParseMode(object, int);
```

COM

```
object.SuiteParseMode = SuiteParseMode
```

SetAliasMode

This function controls how Address Object handles a nickname or former name of street inputs, and if they will be converted to the preferred street name or be preserved.

Remarks

The **SetAliasMode** function accepts one of two set values of the type AliasPreserveMode. These values are:

ConvertAlias (Default)
Nickname or former name street inputs will be converted to their USPS preferred street name during the verification process. This is the default setting.

PreserveAlias
Nickname or former name street inputs will be preserved during the verification process.

Syntax

```
object->SetAliasMode(AliasPreserveMode);
```

C

```
mdAddrSetAliasMode(object, int);
```

COM

```
object.AliasMode = AliasPreserveMode
```

2.3.3: Configure the CASS Form 3553

U.S. Only — The following functions are used to populate user-configurable values required for CASS Form 3553:

SetPS3553_D3_Name

This function sets the name of the list owner. Used to populate item D3 on CASS Form 3553.

Syntax

```
object->SetPS3553_D3_Name(StringValue);
```

C

```
mdAddrSetPS3553_D3_Name(object, char*);
```

COM

```
object.SetPS3553_D3_Name(StringValue)
```

SetPS3553_D3_Company

This function sets the name of the company that either owns the list or for whom the mailing is being prepared. Used to populate item D3 on CASS Form 3553.

Syntax

```
object->SetPS3553_D3_Company(StringValue);
```

C

```
mdAddrSetPS3553_D3_Company(object, char*);
```

COM

```
object.SetPS3553_D3_Company(StringValue)
```

SetPS3553_D3_Address

This function sets the street address of the list owner for CASS Form processing.

Used to populate item D3 on CASS Form 3553.

Syntax

```
object->SetPS3553_D3_Address(StringValue);
```

C

```
mdAddrSetPS3553_D3_Address(object, char*);
```

COM

```
object.SetPS3553_D3_Address(StringValue)
```

SetPS3553_D3_City

This function sets the name of the city from the address of the list owner. Required to populate item D3 on CASS Form 3553.

Syntax

```
object->SetPS3553_D3_City(StringValue);
```

C

```
mdAddrSetPS3553_D3_City(object, char*);
```

COM

```
object.SetPS3553_D3_City(StringValue)
```

SetPS3553_D3_State

This function sets the name of the state from the mailing address of the list owner. Used to populate item D3 on CASS Form 3553.

Syntax

```
object->SetPS3553_D3_State(StringValue);
```

C

```
SetPS3553_D3_State(object, char*);
```

COM

```
object.SetPS3553_D3_State(StringValue)
```

SetPS3553_D3_ZIP

This function sets the ZIP Code from the mailing address of the list owner. Used to populate item D3 on CASS Form 3553.

Syntax

```
object->SetPS3553_D3_ZIP(StringValue);
```

C

```
mdAddrSetPS3553_D3_ZIP(object, char*);
```

COM

```
object.SetPS3553_D3_ZIP(StringValue)
```

SetPS3553_B4_ListName

This function sets the name or identification number of the address list. If more than one list is used, this function is left blank. If the identification number is used, it must be preceded by “ID#.”

Used to generate item B4 on CASS Form 3553.

Syntax

```
object->SetPS3553_B4_ListName(StringValue);
```

C

```
mdAddrSetPS3553_B4_ListName(object, char*);
```

COM

```
object.SetPS3553_B4_ListName(StringValue)
```

SetPS3553_B1_ProcessorName

This function sets the name of the company that coded the address list(s) and/or used the Address Object to perform ZIP + 4 matching. You may attach a list to the generated CASS Form if more space is required.

Used to populate item B1 on CASS Form 3553.

Syntax

```
object->SetPS3553_B1_ProcessorName(StringValue);
```

C

```
mdAddrSetPS3553_B1_ProcessorName(object, char*);
```

COM

```
object.SetPS3553_B1_ProcessorName(StringValue)
```

2.3.4: Configure the SOA Form

Canada Only — The following functions are used to populate and retrieve the Canada Post Summary of Addresses if you are using the Canadian add-on for Address Object:

SetSOACustomerInfo

This function sets the company name and mailing address for use on a Summary of Addresses report.

Remarks

This function must be called before generating the Summary of Addresses with the **GetFormSOA** or **SaveFormSOA** functions.

Input Parameters

The **SOASetCustomerInfo** function accepts two string values as parameters:

CustName	The company name that appears on the mailers Canada Post contract.
CustAddress	The mailing address that appears on the mailers Canada Post contract.

Syntax

```
object->SetSOACustomerInfo(CustName, CustAddress)
```

C

```
mdAddrSetSOACustomerInfo(object, CustName, CustAddress)
```

COM

```
object.SetSOACustomerInfo(CustName, CustAddress)
```

SetSOACPCNumber

This function accepts a string value containing the Canada Post Contract number that must appear on the Summary of Addresses.

Remarks

The CPC Number is an eight-digit number that appears on your Canada Post contract.

Syntax

```
object->SetSOACPCNumber(StringValue);
```

C

```
mdAddrSetSOACPCNumber(object, char*);
```

COM

```
object.SOACPCNumber = StringValue
```

2.3.5: Supply the Address Information to Be Verified

The following functions supply the address information to be verified by a call to the **VerifyAddress** function. After the function call, the return values of these functions will be populated with the verified and standardized address data:

ClearProperties

This function clears all of the stored address information values of the AddressCheck Interface.

Remarks

Call this function before calling the address setting functions with the next address to verify.

Calling this function ensures that the data for each verified address belongs only to that address.

Syntax

```
object->ClearProperties();
```

C

```
mdAddrClearProperties(object);
```

COM

```
object.ClearProperties
```

SetCompany

This function sets the name of the company associated with the input address.

Remarks

The **SetCompany** function accepts a string value containing an optional company name.

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 **VerifyAddress** function will not change the company name supplied by the user.

SuiteLink and the AddressPlus add-on enable Address Object to assign the suite information tied to the Company Name. If a company name is supplied for a company that has been assigned a specific suite, then the return value of the **GetSuite** function will be populated with the suite information.

Syntax

```
object->SetCompany(StringValue);
```

C

```
mdAddrSetCompany(object, char*);
```

COM

```
object.Company = StringValue
```

SetAddress

This function sets the delivery address associated with the input address.

Remarks

This function is required for a successful call to the **VerifyAddress** function.

Syntax

```
object->SetAddress(StringValue);
```

C

```
mdAddrSetAddress(object, char*);
```

COM

```
object.Address = StringValue
```

SetAddress2

Passes a second address line associated with the input address.

Remarks

This function is supplied by the user and is optional. The string value passed to SetAddress2 can contain either a suite or a different, secondary address such as a P.O. Box.

If the **VerifyAddress** function detects a common suite name such as #, APT, STE, and so on, it will append it to the end of the value passed to the **SetAddress** function before attempting to verify the address.

If the address passed via the **SetAddress** function is not verifiable and a separate and complete verifiable address is passed to the **SetAddress2** function, the parameters passed to the **SetAddress** and the **SetAddress2** functions will be swapped and the value originally passed to the **SetAddress2** function will be used to verify the address.

Example #1:

The value passed to the **SetAddress** function was not verifiable, therefore the value passed to the **SetAddress2** function was verified and the values passed to the **SetAddress** and **SetAddress2** functions were swapped.

Input:	SetAddress("123 Main St Apt 10)" (cannot be verified) SetAddress2("PO Box 223") (can be verified)
Output:	GetAddress = "PO Box 223" GetAddress2 = "123 Main St Apt 10"

Example #2:

The value passed to the **SetAddress** function could be verified, so the value passed to **SetAddress2** was not considered.

Input:	SetAddress("PO Box 223") (could be verified) SetAddress2("123 Main St Apt 10") (not looked at)
Output:	GetAddress = "PO Box 223 " GetAddress2 = "123 Main St Apt 10"

Syntax

```
object->SetAddress2(StringValue);
```

C

```
mdAddrSetAddress2(object, char*);
```

COM

```
object.Address2 = StringValue
```

SetSuite

This function sets the suite name and number associated with the input address.

Remarks

If the address being verified contains a suite number, that value can be attached to the end of the street address passed via the **SetAddress** function or passed separately using this function.

Syntax

```
object->SetSuite(StringValue);
```

C

```
mdAddrSetSuite(object, char*);
```

COM

```
object.Suite = StringValue
```

SetCity

This function accepts a string containing the name of the city or municipality associated with the address to be verified.

Remarks

The **SetCity** function is optional if you provide the correct ZIP or Postal Code via the **SetZip** function. If you do not provide a ZIP or Postal Code, the **SetCity** and **SetState** functions are required to successfully check an address.

If the **SetCity** function is not used, the address checking logic will use the official city or municipality name of the ZIP or Postal Code instead.

If the **SetCity** function is supplied, the address checking logic will only change the city or municipality name if it is an incorrect or unapproved mailing name. In these cases, the official city or municipality name for the ZIP or Postal Code will be returned. However, if the official city or municipality name or an approved vanity name for the ZIP or Postal Code is entered, the address checking logic will return that city as entered.

If the supplied city and state do not match the ZIP or Postal Code, the address checking logic will give preference to the city name. Matches will be attempted within the supplied city instead of the ZIP or Postal Code. This logic is based on the

assumption that a ZIP or Postal Code with one typo will result in more incorrect address matches than a city name with a few typos.

Syntax

```
object->SetCity(StringValue);
```

C

```
mdAddrSetCity(object, char*);
```

COM

```
object.City = StringValue
```

SetState

This function sets the two-letter state or province abbreviation associated with the input address.

Remarks

The state is an optional value only when you provide the correct ZIP or Postal Code. If you do not provide a ZIP or Postal Code, the **SetCity** and **SetState** functions will be required to successfully check an address.

Syntax

```
object->SetState(StringValue);
```

C

```
mdAddrSetState(object, char*);
```

COM

```
object.State = StringValue
```

SetZip

This function sets the five-digit or nine-digit U.S. ZIP Code or the six-character Canadian Postal Code associated with the input address. If this function is called, the **SetCity** and **SetZip** functions are optional.

Remarks

The ZIP or Postal Code is only optional when you provide the correct city/municipality and state/province. If you do not provide a city/municipality and state/province, the ZIP Code will be required to successfully check an address.

Syntax

```
object->SetZip(StringValue)
```

C

```
mdAddrSetZip(object, char*)
```

COM

```
object.Zip = StringValue
```

SetPlus4

U.S. Only — This function sets the four-digit ZIP Code add-on associated with the input address.

Remarks

This function is optional. When correcting addresses, it is usually ignored. However, if the input ZIP Code is a “unique” ZIP Code (a ZIP Code that is assigned to one company), the input Plus4 code will be retained under certain circumstances.

If the user does not supply the Plus4, the address checking logic will still code the address, but it may use a “default” Plus4 code for these “unique” ZIP codes.

Syntax

```
object->SetPlus4(StringValue);
```

C

```
mdAddrSetPlus4(object, char*);
```

COM

```
object.Plus4 = StringValue
```

SetLastLine

This function sets the city, state and ZIP Code combination from a single string value.

Remarks

You can use this function as an alternative function to calling the **SetCity**, **SetState**, and **SetZip** functions individually. For example, pass “Rancho Santa Margarita, CA 92688-2212” to SetLastLine rather than passing “Rancho Santa Margarita” to the **SetCity** function, “CA” to the **SetState** function, and “92668” to the **SetZip** function.

This function is useful if you have address data that is composed of lines of plain text and the City, State, and ZIP codes do not have their own separate fields.

Syntax

```
void = object->SetLastLine(StringValue);
```

C

```
void = mdAddrSetLastLine(object, char*);
```

COM

```
object.LastLine = StringValue
```

SetCountryCode

This function sets a two-character abbreviation that indicates the country associated with the input address. The value passed to this function can optionally be included with the input address.

Remarks

Address Object includes data for American and Canadian addresses, so values other than “US” or “CA” for Canada are ignored.

Syntax

```
void = object->SetCountryCode(StringValue);
```

C

```
void = mdAddrSetCountryCode(object, char*);
```

COM

```
object.CountryCode = StringValue
```

SetUrbanization

Puerto Rico Only — This function sets an urbanization name associated with the input address. This function applies to Puerto Rican addresses only.

Remarks

The **SetUrbanization** function is only used when attempting to correct addresses in Puerto Rico. If it is not called, 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

```
void = object->SetUrbanization(StringValue);
```

C

```
void = mdAddrSetUrbanization(object, char*);
```

COM

```
object.Urbanization = StringValue
```

2.3.6: Other Input Functions

The following functions are not always used for address verification. The **SetLastName** function is used if you are using Address Object with the AddressPlus option. The parsed address functions can be used in place of the **SetAddress**, **SetAddress2**, and **SetSuite** functions if your address information is already parsed:

SetLastName

This function sets the last name associated with a residential address. This function is required to use the AddressPlus add-on.

Remarks

AddressPlus is an add-on that enables Address Object to assign suite numbers for some residential addresses. In addition to the basic address information (**SetAddress**, **SetCity**, **SetState**, and **SetZip** functions), you must also submit a correct last name associated with the address record to enable Address Object to distinguish the record from other suites at the same address.

Syntax

```
void = object->SetLastName(StringValue);
```

C

```
void = mdAddrSetLastName(object, char*);
```

COM

```
object.LastName = StringValue
```

SetParsedAddressRange

This function sets the delivery number of a street address.

Remarks

This function accepts a 10-character maximum string value. It can be used to pass the range portion of an address that has already been parsed instead of using the **SetAddress** function.

Syntax

```
void = object->SetParsedAddressRange(StringValue);
```

C

```
void = mdAddrSetParsedAddressRange(object, char*);
```

COM

```
object.ParsedAddressRange = StringValue
```

SetParsedPreDirection

This function sets the directional that precedes the street name of a submitted street address.

Remarks

This function accepts a two-character maximum string value to pass the pre-directional portion of a parsed address to the AddressCheck Interface.

Syntax

```
void = object->SetParsedPreDirection(StringValue);
```

C

```
void = mdAddrSetParsedPreDirection(object,string);
```

COM

```
object.ParsedPreDirection = StringValue
```

SetParsedStreetName

This function sets the street name portion of an address.

Remarks

This function accepts a maximum 28-character string containing the street name portion of an address to be verified.

Syntax

```
void = object->SetParsedStreetName(StringValue);
```

C

```
void = mdAddrSetParsedStreetName(object, char*);
```

COM

```
object.ParsedStreetName = StringValue
```

SetParsedSuffix

This function sets the street suffix portion of an address.

Remarks

This function accepts a four-character maximum string value containing the suffix portion of a street address to be verified.

Typical suffix values might include: “ST;” “RD;” “AVE;” “BLVD;” “CIR;” and “PL”.

Syntax

```
void = object->SetParsedSuffix(StringValue);
```

C

```
void = mdAddrSetParsedSuffix(object, char*);
```

COM

```
object.ParsedSuffix = StringValue
```

SetParsedPostDirection

This function sets any directional that follows the street name of a submitted street address.

Remarks

This function accepts a two-character maximum string value to pass the post-directional portion of an already parsed address to the AddressCheck Interface.

Possible values which can be passed to this function are: “N,” “NE,” “E,” “SE,” “S,” “SW,” “W,” and “NW.”

Syntax

```
void = object->SetParsedPostDirection(StringValue)
```

C

```
void = mdAddrSetParsedPostDirection(object, char*);
```

COM

```
object.ParsedPostDirection = StringValue
```

SetParsedSuiteName

This function sets the name of the secondary unit of an address.

Remarks

This function accepts a four-character maximum string value containing the name portion of a secondary address.

Possible values that can be passed to this function are:

"#;" "APT;" "BLDG;" "BOX;" "BSMT;" "DEPT;" "FL;" "FRNT;" "HNGR;" "LBBY;" "LOT;" "LOWR;" "OFC;" "PH" (Penthouse); "PIER;" "REAR;" "RM;" "SIDE;" "SLIP;" "SPC;" "STE;" "STOP;" "TRLR;" "UNIT;" "UPPR."

Syntax

```
void = object->SetParsedSuiteName(StringValue);
```

C

```
void = mdAddrSetParsedSuiteName(object, char*);
```

COM

```
object.ParsedSuiteName = StringValue
```

SetParsedSuiteRange

This function sets the range of any secondary unit of an address.

Remarks

When used to pass the suite range to the **VerifyAddress** function, this function allows you to easily use AddressCheck if your data is already stored in parsed format.

Syntax

```
void = object->SetParsedSuiteRange(StringValue);
```

C

```
void = mdAddrSetParsedSuiteRange(object, char*);
```

COM

```
object.ParsedSuiteRange = StringValue
```

SetParsedRouteService

Canada Only — This function sets the route service information from a parsed Canadian street address.

Remarks

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

Syntax

```
void = object->SetParsedRouteService(StringValue);
```

C

```
void = mdAddrSetParsedRouteService(object, char*);
```

COM

```
object.ParsedRouteService = StringValue
```

SetParsedLockBox

Canada Only — This function sets the Lock Box information from a parsed Canadian street address.

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

```
void = object->SetParsedLockBox(StringValue);
```

C

```
void = mdAddrSetParsedLockBox(object, char*);
```

COM

```
object.ParsedLockBox = StringValue
```

SetParsedDeliveryInstallation

Canada Only — This function sets the Delivery Installation information from a parsed Canadian street address.

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

```
void = object->SetParsedDeliveryInstallation(StringValue);
```

C

```
void = mdAddrSetParsedDeliveryInstallation(object, char*);
```

COM

```
object.ParsedDeliveryInstallation = StringValue
```

2.3.7: Verify the Address Data

The **VerifyAddress** function standardizes and verifies the submitted address information.

VerifyAddress

This function will verify, standardize, and enhance a U.S. or Canadian address based on the input data. You must call the functions listed below with the address to be verified.

Remarks

A TRUE return will set most or all of the functions of the AddressCheck Interface with standardized data. Check the **GetResults** function to determine the level of matching found for the input address.

On a FALSE return, also check the **GetResults** function to determine the cause of the problem.

*If the value passed to the **SetCASSEnable** function is FALSE, this function will only verify and code U.S. addresses to the ZIP + 4 level and will not perform DPV processing. In order to generate a CASS Form 3553 and receive postal discounts, you must set **SetCASSEnable** to TRUE.*

Input Functions

The **VerifyAddress** function requires that you call some, or all, of the following functions:

SetCompany	Optional
SetAddress	Required
SetSuite	Optional
SetCity	Optional or Required (see below)
SetState	Optional or Required (see below)
SetZip	Optional or Required (see below)
SetLastLine	Optional or Required (see below)
SetPlus4	U.S. Only & Optional
SetUrbanization	U.S. Only & Optional
SetCountryCode	Optional

The city and state are optional if the ZIP Code is provided and the ZIP Code is optional if the city and state are provided.

If the city, state and ZIP Code information is contained in a single field, the ***SetLastLine*** *function can be used in place of the City, State and ZIP.*

Return Value

The **VerifyAddress** function returns a Boolean value of 0 (FALSE) if the address cannot be coded, or 1 (TRUE) if the address can be coded.

Call the **GetResults** function after this to determine the match level with the national database. On a fully verified address (results code “AS01”), Address Object will populate return values for most, if not all, of the functions listed below, when applicable to the submitted address.

When the input address is a non-USPS address in the U.S. (results code “AS03”), the following functions will **not** return a value:

GetAddressTypeCode	GetAddressTypeString
GetCMRA	GetCarrierRoute
GetELotNumber	GetELotOrder
GetDPVFootnotes	GetEWSCount
GetEWSFlag	GetLacs
GetLACSLinkIndicator	GetLACSLinkReturnCode
GetSuiteLinkReturnCode	GetSuiteStatus

When verifying a Canadian address using the Canadian add-on, only the following functions will return a value, if applicable:

GetResults	GetCompany
GetAddress	GetAddress2
GetSuite	GetCity
GetState	GetZip
GetParsedAddressRange	GetParsedPreDirection
GetParsedStreetName	GetParsedPostDirection
GetParsedSuffix	GetParsedSuiteName
GetParsedSuiteRange	GetParsedRouteService
GetParsedLockBox	GetParsedDeliveryInstallation
GetParsedGarbage	

When using the RBDI add-on, the **GetRBDI** function will return a value for U.S. addresses.

Syntax

```
BooleanValue = object->VerifyAddress();
```

C

```
IntegerValue = mdAddrVerifyAddress(object);
```

COM

```
BooleanValue = object.VerifyAddress()
```

2.3.8: Retrieve Status and Return Codes

The following functions return information about the level and types of data matches found by a call to the **VerifyAddress** function:

GetResults

This function returns a comma-delimited string of four-character codes that detail the level of matching found for the current address, any errors that occurred during the last call to the **VerifyAddress** function, and any changes that were made in the process of standardizing the information.

Remarks

The **GetResults** function is intended to replace the **GetStatusCode**, **GetErrorCode**, **GetErrorString**, **GetDPVFootnotes**, and **GetSuiteStatus** functions, providing a single source of information about the last **VerifyAddress** function call and eliminating the need to call multiple functions to determine if a particular address was fully coded and verified.

The function returns one or more of the following codes in a comma-delimited list:

Code	Short Description	Long Description
		Status Results 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.
AS11	PBSA Address	A Post Office Box address with a street style address instead of the normal PO Box # address format.
AS13	Address Updated By LACS	US Only. The address has been converted by LACSLink® from a rural-style address to a city-style address.
AS14	Suite Appended	US Only. A suite was appended by SuiteLink™ using the address and company name.

Code	Short Description	Long Description
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).
AS22	No Alternate Address Suggestion Found	No alternate address suggestion was found for this address.
AS23	Extraneous Information	Extraneous information not used in verifying the address was found. This includes unnecessary sub premises and other unrecognized data. The unrecognized data has been placed in the ParsedGarbage field.
AS24	USPS Door Not Accessible	Address identified by the USPS where carriers cannot physically access a building or door for mail delivery.
		Error Results 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.
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.

Code	Short Description	Long Description
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 number 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.
AE15	Demo Mode	Limited to Demo Mode operation. Please contact a Melissa Data CSR at 1-800-800-6245 x4.
AE16	Expired Database	The Database has expired. Please contact a Melissa Data CSR at 1-800-800-6245 x4.
AE17	Sub Premise Not Required	This result code is no longer returned. Unnecessary sub premises will now return AS23.
AE19	Find Suggestion Timeout	The FindSuggestion function has exceeded the time limit.
AE20	Find Suggestion Disabled	The FindSuggestion function is disabled. See the manual for further details.
		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.
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.

Code	Short Description	Long Description
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.

These results codes are designed to clearly indicate 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 use results codes to filter 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 results code “AS20.”

Change codes do not indicate a problem with the validity of the address, but that Address Object made minor changes to accurize and standardize the information.

For more information on coding for results codes, see *Using Results Codes: Coding for the present and the future* on page 4.

Syntax

```
StringValue = object->GetResults();
```

C

```
StringValue = mdAddrGetResults(object);
```

COM

```
StringValue = object.Results
```

GetStatusCode (Deprecated)

This function returns a status code that indicates the level of coding that was performed on the input address.

This function has been deprecated. You should use the **GetResults** function instead. See page 65 for documentation on this function.

Remarks

The **GetStatusCode** function returns a one-character value set by a call to the **VerifyAddress** function.

The status codes and their related coding levels are as follows:

Code	Level
D	Demo Mode
E	Expired database
S (U.S.)	The address was standardized but not coded. Standardization means that some conversion was done on the address (for example, changing Post Office Box to PO Box or abbreviating street suffixes).
V	Street number validated to DPV level.
F	DPV processing was terminated due to the detection of what is determined to be an artificially created address. No address beyond this point has been DPV validated. In accordance with the License Agreement between USPS and Melissa Data, DPV shall be used to validate legitimately obtained addresses only and shall not be used for the purpose of artificially creating address lists. The written Agreement between Melissa Data and you, its customer, shall also include this same restriction against using DPV to artificially create address lists. Continuing use of DPV requires compliance with all terms of the License Agreement. If you believe this address was identified in error, please contact Melissa Data.
X	Address was not coded.
6	A Canadian address was fully coded.
7 (U.S.)	There were multiple matches for the address but they were all in the same ZIP Code and carrier route. The returned ZIP Code and carrier route will be correct but you will not get any ZIP + 4 information.
9 (U.S.)	The address was fully coded.

If an ‘S’ or ‘X’ code is returned, check the **GetErrorCode** function to find out why the address did not code.

The return value from **GetStatusCode** refers to the level of coding achieved for the street number of a given address. To check the level of coding achieved for the suite/apartment number of a given address, check the **GetSuiteStatusCode** function (or **GetDPVFootnotes**).

This function is deprecated. You should begin using the **GetResults** function as soon as possible.

Syntax

```
StringValue = object->GetStatusCode();
```

C

```
StringValue = mdAddrGetStatusCode(object);
```

COM

```
StringValue = object.StatusCode
```

GetErrorCode (Deprecated)

Deprecated. This function returns a one-letter code to describe any problems caused by an attempt to verify the input address.

This function has been deprecated. You should use the **GetResults** function instead. See page 65 for documentation on this function.

Remarks

The **GetErrorCode** function returns a one-character value that describes any problems encountered if an address cannot be verified.

This field will always contain a letter when the **VerifyAddress** function returns a “0” (FALSE) indicating that the address could not be verified.

The error codes and their related error types are listed on page 71.

This function is deprecated. You should begin using the **GetResults** function as soon as possible.

Syntax

```
StringValue = object->GetErrorCode();
```

C

```
StringValue = mdAddrGetErrorCode(object);
```

COM

```
StringValue = object.ErrorCode
```

GetErrorString (Deprecated)

This function returns a string value which describes the meaning of the return value from the **GetErrorCode** function in greater detail.

> **This function has been deprecated.** You should use the **GetResults** function instead. See page 65 for documentation on this function.

Remarks

The **GetErrorString** function describes any problems encountered if an address cannot be verified.

This function will always return a descriptive error string when the **VerifyAddress** function returns a “0” (FALSE) indicating that the address could not be verified.

The error codes and their related error types are listed below.

This function is deprecated. You should begin using the **GetResults** function as soon as possible.

Syntax

```
StringValue = object->GetErrorString();
```

C

```
StringValue = mdAddrGetErrorString(object);
```

COM

```
StringValue = object.ErrorString
```

Error Codes and Strings

The following table shows the possible values returned by the **GetErrorCode** and **GetErrorString** functions.

Code	ErrorString Returned	Details
M	Multiple Matches	More than one record matches the address and there is not enough information available in the input address to break the tie between multiple records. Passing information, such as city/municipality names or urbanization names, can help reduce the number of multiple match errors.
R	Range Error	The address was found but the street number in the input address was not between the low and high range of the Post Office database.
S	Invalid Suite (Canadian address)	(Canadian Addresses Only). The suite was missing or not correct. Canadian addresses cannot be coded to a default site address as they can be in the U.S.

Code	ErrorString Returned	Details
T	Component Error	Either the directionals or the suffix field did not match the post office database, or there was more than one choice for correcting the address. For example, if the given address was “100 Main St” and the only addresses found were “100 E Main St” and “100 Main Ave,” the error code “T” would be returned because we do not know whether to add the directional “E” or to change the suffix to “Ave.”
U	Unknown Street	An exact street name match could not be found and phonetically matching the street name resulted in either no matches or matches to more than one street name.
X	Undeliverable Address	U.S. Addresses only. The physical location exists but there are no homes on this street. One reason might be railroad tracks or rivers running alongside this street, as they would prevent construction of homes in this location.[1]
Z	Invalid ZIP Code	The Postal Code does not exist and could not be determined by the city/municipality and state/province.
C	Canadian ZIP Code	A Canadian Postal Code was passed to the **VerifyAddress** function and either the **SetPathToCanadaFiles** function was not called or the current license string only permits Address Object to code U.S. addresses.
F	Foreign Address	The submitted postal code was from a nationality for which the data files were not initialized.[2]
E	Database Expired	The current files have expired.
D	Demo Mode Only	While in Demo mode, you can only verify Nevada addresses.
W	Early Warning Address	This address has been identified in the Early Warning System (EWS) data file and should be included in the next national database update.[3]
2	Address2 Coded	Address Object could not code the parameter passed to the **SetAddress** function and the value passed to the **SetAddress2** function was coded instead.[4]
Empty	No Error	Address is correct

1. Undeliverable U.S. address only.
2. For example, the submitted postal code was Canadian and the **SetPathToCanadaFiles** function was not called.
3. This code is only returned when **EnableEarlyWarningSystem** function has been called.
4. This code is returned if the **DisableAddressSwapping** function is set to TRUE. Since this function is only present in the COM Object interface, this error code will never be returned by the cross-platform Address Object.

GetDPVFootnotes

U.S. Only — This function returns the applicable standard USPS footnote codes for the input address.

Remarks

This function will return a string value up to 6 characters long containing the applicable DPV footnote codes.

The following standard footnotes are available:

Code	Description
A1	Input Address Not Matched to the ZIP + 4 file.
AA	Input Address Matched to the ZIP + 4 file.
BB	Input Address Matched to DPV (all components).
C1	Input address primary number matched, secondary number not matched; secondary number required.
CC	Input address primary number matched, secondary number not matched, secondary number not required.
F1	Address Was Coded to a Military Address.
G1	Address Was Coded to a General Delivery Address.
IA	Informed Address identified.
M1	Input Address Primary Number Missing.
M3	Input Address Primary Number Invalid.
N1	Input address primary number matched to DPV but address missing required secondary number.
P1	Input Address Missing PO, RR, or HC Box number.
P3	Input Address PO, RR, or HC number invalid.
PB	Identified PO Box Street Address.
R1	Input Address Matched to CMRA but Secondary Number not Present.
R7	Addresses that are assigned to a phantom route of R777 or R779.
RR	Input Address Matched to CMRA.
TA	Input address primary number matched to DPV by dropping trailing alpha.
U1	Address Was Coded to a Unique ZIP Code.

Syntax

```
StringValue = object->GetDPVFootnotes();
```

C

```
StringValue = mdAddrGetDPVFootnotes(object);
```

COM

```
StringValue = object.DPVFootnotes
```

GetEWSCount

U.S. Only — The return value of this function increments by one for each EWS record found by the current instance of Address Object.

Remarks

The AddressCheck Interface keeps a count of EWS records encountered by the current instance of Address Object (for CASS reporting purposes only) and returns it via this function

On the COM version of Address Object, the return value of this function will only be accurate if the**EnableEarlyWarningSystem** function has been successfully called prior to the first call the **VerifyAddress** function. Regardless of the version in use, the return value will only be accurate if ews.txt is in the data files folder for the cross-platform version of Address Object.

Syntax

```
IntegerValue = object->GetEWSCount();
```

C

```
IntegerValue = mdAddrGetEWSCount(object);
```

COM

```
IntegerValue = object.EWSCount
```

GetEWSFlag

U.S. Only — This function returns a one-character string value which indicates whether the current address record was found in the Early Warning System (EWS) database.

Remarks

The **GetEWSFlag** function returns a "Y" if the current address was found in the EWS database. The **GetResults** function will also return a "AE06."

This function will only return a value if a TRUE value was passed to the **EnableEarlyWarningSystem** function.

On the COM version of Address Object, this function will only return a value if the **EnableEarlyWarningSystem** function has been successfully called prior to the first call to the **VerifyAddress** function. Regardless of the version in use, the function will only return a value if ews.txt is in the data files folder for the cross-platform version of Address Object.

Syntax

```
StringValue = object->GetEWSFlag();
```

C

```
StringValue = mdAddrGetEWSFlag(object);
```

COM

```
StringValue = object.EWSFlag
```

GetLacs

*If you are using the LACS^{Link} feature of Address Object to update addresses, you should use the **GetLACSLinkIndicator** function instead of this one.*

U.S. Only — This function returns a one-character string which indicates if the input address has undergone a conversion to a city-style street address.

Remarks

The **GetLacs** function returns a one-character value set by a call to the **VerifyAddress** function.

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

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

This function does not indicate that Address Object updated the address. For that information, check the **GetLACSLinkIndicator** function.

Syntax

```
StringValue = object->GetLACS();
```

C

```
StringValue = mdAddrGetLACS(object);
```

COM

```
StringValue = object.LACS
```

GetLACSLinkIndicator

U.S. Only — This function returns a "Y" if the submitted address was corrected to a new address found in the LACS^{Link} data. An "S" is returned if the submitted address was corrected to a new address found in the LACS^{Link} data but contained a suite that could not be matched. An "N" is returned if the address was not updated by LACS^{Link}.

Remarks

LACS^{Link} is a process where some rural route addresses are modified to city-style addresses to allow emergency services (for example, ambulance, police, fire, and so on) to find these addresses more efficiently.

The LACS^{Link} service matches the old address with the updated address and corrects it as part of the Address Check process.

If the submitted address was found in the LACS^{Link} database and changed to the new address, this function will return a "Y," otherwise it will return an "N." An "S" if the submitted address was corrected to a new address found in the LACS^{Link} data but contained a suite that could not be matched.

This indicator is only returned for LACS addresses (rural route addresses whether or not they have been converted).

Syntax

```
StringValue = object->GetLACSLinkIndicator();
```

C

```
StringValue = mdAddrGetLACSLinkIndicator(object);
```

COM

```
StringValue = object.LACSLinkIndicator
```

GetLACSLinkReturnCode

U.S. Only — This function returns a two-character number code indicating the degree to which the submitted address was matched to the LACS^{Link} data and if the address was updated.

Remarks

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

The LACS^{Link} service matches the old address with the updated address and corrects it as part of the Address Check process.

This function returns one of the following codes:

Code	Description
A	**LACS Record Match** - The input record matched to a record in the master file. A new address could be furnished.
00	**No Match** - The input record *could not be* matched to a record in the master file. A new address could not be furnished.
14	**Found LACS Record: New Address Would Not Convert at Run Time** - The input record matched to a record in the master file. The new address could not be converted to a deliverable address.
92	**LACS Record: 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 or high-rise match.

This return code is only returned for LACS addresses (rural route addresses whether or not they have been converted).

Syntax

```
StringValue = object->GetLACSLinkReturnCode();
```

C

```
StringValue = mdAddrGetLACSLinkReturnCode(object);
```

COM

```
StringValue = object.LACSLinkReturnCode
```

GetRBDI

For information about purchasing the RBDI add-on for Address Object, call 1-800-MELISSA.

U.S. Only — This function returns a one-character code indicating whether the submitted address is a residence, a business, or the status is unknown.

Requires

RBDI add-on for Address Object.

Remarks

Parcel delivery to residential addresses is more expensive than to business addresses. You can use the return value of this function to filter out one type of address or the other, or determine the best carrier for the delivery.

If the RBDI add-on is installed, this function will return one of the following codes:

Code	Definition
R	Residential Address
B	Business Address
U	Unknown

Syntax

```
StringValue = object->GetRBDI();
```

C

```
StringValue = mdAddrGetRBDI(object);
```

COM

```
StringValue = object.RBDI
```

GetSuiteLinkReturnCode

U.S. Only — This function returns a string value indicating whether a match was found for the submitted address record in the SuiteLink data file.

Remarks

SuiteLink is a product offered by the U.S. Postal Service which enables Address Object to link a business address to a suite number and update your address records that are missing suite information.

The SuiteLink Return Code indicates whether or not the submitted address was matched to the database and an improved suite number could be returned.

Code	Description
A	The input record matched to a record in the master file. An improved business address could be furnished.
00	The input record could not be matched to a record in the master file. An improved business address could not be furnished.

If this function returns a string value of “A,” an improved suite number will be returned by the **GetSuite** function.

Syntax

```
StringValue = object->GetSuiteLinkReturnCode();
```

C

```
StringValue = mdAddrGetSuiteLinkReturnCode(object);
```

COM

```
StringValue = object.SuiteLinkReturnCode
```

GetSuiteStatus

U.S. Only — Deprecated. This function returns a status code which indicates the success or failure to validate the suite/apartment number of a submitted address. If DPV checking is enabled, this value reflects whether or not the given suite number is an actual delivery point. If DPV checking is not being used, this value reflects whether or not the given suite number falls within the valid range of suite numbers for the address submitted.

Remarks

The **GetSuiteStatus** function returns a one-character value set by a call to the **VerifyAddress** function.

The status codes and their related coding levels are as follows:

Code	Level
M	A suite number is required for the given street address but is missing from the submitted record.
R	A suite number is present on the submitted record but is either not required or is out of range for the given street address.
V	The suite field was verified.
X	The suite field was not coded.

This function is deprecated. You should begin using the **GetResults** function as soon as possible.

Syntax

```
StringValue = object->GetSuiteStatus();
```

C

```
StringValue = mdAddrGetSuiteStatus(object);
```

COM

```
StringValue = object.SuiteStatus
```

2.3.9: Retrieve the Standardized and Verified Address Data

The following functions retrieve the address information that has been verified and standardized by a call to the **VerifyAddress** function:

GetCompany

This function returns the name of the company associated with the input address.

Remarks

The **VerifyAddress** function will not change the company name supplied by the user.

Syntax

```
StringValue = object->GetCompany();
```

C

```
StringValue = mdAddrGetCompany(object);
```

COM

```
StringValue = object.Company
```

GetAddress

This function returns the standardized delivery address associated with the input address.

Remarks

After a call to the **VerifyAddress** function, the **GetAddress** function will return the address passed to the **SetAddress** function, including any corrections or standardizations performed by the address checking logic.

Address corrections may include fixing misspelled street names or inserting missing suffixes and directionals.

Address standardization involves the conversion of suffixes and directionals to preferred postal abbreviations (for example, “Street” to “St”).

*If a suite name is attached to the end of the parameter passed to **SetAddress**, it will be returned by **GetSuite** function. If the address is a private mailbox, the PMB will be returned by the **GetPrivateMailbox** function.*

*To return the full address in one line, combine the return values of the **GetAddress**, **GetSuite**, and **GetPrivateMailbox** functions into a single string value.*

Syntax

```
StringValue = object->GetAddress();
```

C

```
StringValue = mdAddrGetAddress(object);
```

COM

```
StringValue = object.Address
```

GetAddress2

This function returns the second address line associated with the input address.

Remarks

This function normally returns the unverified string value passed to the **SetAddress2** function. However, if the value passed to the **SetAddress** is not verifiable and the value passed to **SetAddress2** contains a verifiable address, this function will return the unverified value originally passed to **SetAddress**.

Syntax

```
StringValue = object->GetAddress2();
```

C

```
StringValue = mdAddrGetAddress2(object);
```

COM

```
StringValue = object.Address2
```

GetSuite

This function returns the suite name and number associated with the input address.

Remarks

If the suite number is part of the value passed to the **SetAddress** function, it will be returned via the **GetSuite** function. This move helps maintain a clean database and allows the user to limit secondary ranges to a separate field.

If you want the address and suite combined, you will need to join the return values of these two functions together. Make sure to add a space between the values when you join them.

Syntax

```
StringValue = object->GetSuite();
```

C

```
StringValue = mdAddrGetSuite(object);
```

COM

```
StringValue = object.Suite
```

GetCity

After a successful call to the **VerifyAddress** function, this function returns a string value containing the name of the city or municipality associated with the verified address.

Remarks

If the city name was not supplied before calling **VerifyAddress**, the address checking logic will return the official city or municipality name of the ZIP or Postal Code instead.

If the **SetCity** function was called, the address checking logic will only change the city or municipality name if it is an incorrect or unapproved mailing name. In these cases, the official city or municipality name for the ZIP or Postal Code will be returned. However, if the official city or municipality name or an approved vanity name for the ZIP or Postal Code is entered, the address checking logic will return that city as entered.

If the supplied city and state do not match the ZIP or Postal Code, the address checking logic will give preference to the city name. Matches will be attempted within the supplied city instead of the ZIP or Postal Code. This logic is based on the

assumption that a ZIP or Postal Code with one typo will result in more incorrect address matches than a city name with a few typos.

Syntax

```
StringValue = object->GetCity();
```

C

```
StringValue = mdAddrGetCity(object);
```

COM

```
StringValue = object.City
```

GetState

This function returns the two-letter state or province abbreviation associated with the standardized address.

Remarks

If the value passed to the **SetState** function is incorrect or is not supplied, the address checking logic will return the correct two-letter state abbreviation to this function if the submitted ZIP or Postal Code is valid.

Syntax

```
StringValue = object->GetState();
```

C

```
StringValue = mdAddrGetState(object);
```

COM

```
StringValue = object.State
```

GetZip

This function returns the five-digit or nine-digit U.S. ZIP Code or the six-character Canadian Postal Code associated with the input address. It is always populated by a successful call to the **VerifyAddress** function.

Remarks

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

Syntax

```
StringValue = object->GetZip()
```

C

```
StringValue = mdAddrGetZip(object)
```

COM

```
StringValue = object.Zip
```

GetPlus4

U.S. Only — This function returns the four-digit ZIP Code add-on associated with the input address.

Remarks

If the user does not pass a value to the **SetPlus4** function, the address checking logic will only code the address with a ZIP + 4 if the address can be fully DPV verified or the **SetCASSEnable** function is set to FALSE and the address can be ZIP + 4 verified.

This function does not return any data when the GetResults function returns the results code “AS03” (Non-USPS address).

Syntax

```
StringValue = object->GetPlus4();
```

C

```
StringValue = mdAddrGetPlus4(object);
```

COM

```
StringValue = object.Plus4
```

GetCountryCode

This function returns a two-character abbreviation that indicates the country associated with the input address.

Remarks

This function returns a string value after a successful call to the **VerifyAddress** function.

Address Object includes data for American and Canadian addresses. “US” is returned for addresses in the United States and “CA” is returned for Canadian addresses.

Syntax

```
StringValue = object->GetCountryCode();
```

C

```
StringValue = mdAddrGetCountryCode(object);
```

COM

```
StringValue = object.CountryCode
```

GetUrbanization

Puerto Rican Addresses Only — This function returns the urbanization name associated with the input address. This function applies to Puerto Rican addresses only.

Remarks

See the details on the **SetUrbanization** function on page 54 to learn how the urbanization is used with Puerto Rican addresses.

Syntax

```
StringValue = object->GetUrbanization();
```

C

```
StringValue = mdAddrGetUrbanization(object);
```

COM

```
StringValue = object.Urbanization
```

2.3.10: Retrieve Parsed Street Address

In addition to verifying and standardizing an address, the AddressCheck Interface will also parse the street address and return the parts via the following functions:

GetParsedAddressRange

This function returns the delivery number of a street address.

Remarks

This function returns the range portion of a street address after a successful call to the **VerifyAddress** function.

Syntax

```
StringValue = object->GetParsedAddressRange();
```

C

```
StringValue = mdAddrGetParsedAddressRange(object);
```

COM

```
StringValue = object.ParsedAddressRange
```

GetParsedPreDirection

This function returns the geographic directional that precedes the street name of a submitted street address.

Remarks

This function returns the pre-directional part of a street address after a successful call to the **VerifyAddress** function. The return value of this function will contain the post-direction of the full address string passed into the **SetAddress** or **SetAddress2** function. The format will depend on the setting of the **SetStandardizationType**

function. “ShortFormat” will abbreviate the directional, “LongFormat” will spell it out, and “AutoFormat” will preserve the original style.

Syntax

```
StringValue = object->GetParsedPreDirection();
```

C

```
StringValue = mdAddrGetParsedPreDirection(object);
```

COM

```
StringValue = object.ParsedPreDirection
```

GetParsedStreetName

This function returns the street name portion of an address.

Remarks

This function returns the street name of the full address string passed via the **SetAddress** or **SetAddress2** functions.

For street names that begin with one of the following Spanish words, that first word will be moved to the suffix field and the next word in the street name will be returned as the street name. This is because Spanish street names have the suffix at the front of the address. If the suffix was not moved to the suffix field, it would appear that every street in some Puerto Rican ZIP codes began with the same name. The Spanish words that get moved to the suffix field are: “Avenida,” “Calle,” “Camino,” “Paseo,” and “Via”

Syntax

```
StringValue = object->GetParsedStreetName();
```

C

```
StringValue = mdAddrGetParsedStreetName(object);
```

COM

```
StringValue = object.ParsedStreetName
```

GetParsedPostDirection

This function returns any geographic directional that follows the street name of a submitted street address.

Remarks

This function will return the post-directional part of a street address after a successful call to the **VerifyAddress** function.

The return value of this function will contain the post-directional part of the full address string passed into the **SetAddress** or **SetAddress2** function. The format will depend on the setting of the **SetStandardizationType** function. “ShortFormat” will abbreviate the directional, “LongFormat” will spell it out, and “AutoFormat” will preserve the original style.

Syntax

```
StringValue = object->GetParsedPostDirection()
```

C

```
StringValue = mdAddrGetParsedPostDirection(object);
```

COM

```
StringValue = object.ParsedPostDirection
```

GetParsedSuffix

This function returns the street suffix portion of a street address.

Remarks

This function returns the suffix of the full street address passed via the **SetAddress** or **SetAddress2** functions.

The output of this function depends on the setting passed to the **SetStandardizationType** function (see page 38).

If the setting is “ShortFormat,” full words are abbreviated. For instance, if the suffix “Street” is found in the input string, it will be converted to “St” before being returned by this function. If the setting is “LongFormat,” the long form of the suffix (“Street”) will be

returned, even if abbreviated in the input door. If the setting is “AutoFormat,” the original form of the suffix in the input street will be preserved.

Syntax

```
StringValue = object->GetParsedSuffix();
```

C

```
StringValue = mdAddrGetParsedSuffix(object);
```

COM

```
StringValue = object.ParsedSuffix
```

GetParsedSuiteName

This function returns the name of the secondary unit of an address.

Remarks

This function returns a string value containing the suite name portion of a street address after a successful call to the **VerifyAddress** function.

Possible return values are:

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

Syntax

```
StringValue = object->GetParsedSuiteName();
```

C

```
StringValue = mdAddrGetParsedSuiteName(object);
```

COM

```
StringValue = object.ParsedSuiteName
```

GetParsedSuiteRange

This function returns the range of any secondary unit of an address.

Remarks

This function returns the suite number portion of a full address string passed via the **SetAddress** or **SetAddress2** functions after a successful call to the **VerifyAddress** function.

Syntax

```
StringValue = object->GetParsedSuiteRange();
```

C

```
StringValue = mdAddrGetParsedSuiteRange(object);
```

COM

```
StringValue = object.ParsedSuiteRange
```

GetParsedPrivateMailboxName

This function returns the name of the private mail box associated with a CMRA (Commercial Mail Receiving Agency).

Remarks

This function returns a three-character maximum string value containing the private mail box name of the full address string after a successful call to the **VerifyAddress** function.

The possible return values for this function are “PMB” and “#.”

Syntax

```
StringValue = object->GetParsedPrivateMailboxName();
```

C

```
StringValue = mdAddrGetParsedPrivateMailboxName(object);
```

COM

```
StringValue = object.ParsedPrivateMailboxName
```

GetParsedPrivateMailboxNumber

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

Remarks

This function returns a 10-character maximum string value containing the private mail box number portion of a street address after a successful call to the **VerifyAddress** function.

Syntax

```
StringValue = object->GetParsedPrivateMailboxNumber();
```

C

```
StringValue = mdAddrGetParsedPrivateMailboxNumber(object);
```

COM

```
StringValue = object.ParsedPrivateMailboxNumber
```

GetParsedRouteService

Canada Only — This function returns the route service information from a parsed Canadian street address.

Remarks

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

Syntax

```
StringValue = object->GetParsedRouteService();
```

C

```
StringValue = mdAddrGetParsedRouteService(object);
```

COM

```
StringValue = object.ParsedRouteService
```

GetParsedLockBox

Canada Only — This function returns the Lock Box information from a parsed Canadian street address.

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

```
StringValue = object->GetParsedLockBox();
```

C

```
StringValue = mdAddrGetParsedLockBox(object);
```

COM

```
StringValue = object.ParsedLockBox
```

GetParsedDeliveryInstallation

Canada Only — This function returns the Delivery Installation information from a parsed Canadian street address.

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

```
StringValue = object->GetParsedDeliveryInstallation();
```

C

```
StringValue = mdAddrGetParsedDeliveryInstallation(object);
```

COM

```
StringValue = object.ParsedDeliveryInstallation
```

GetParsedGarbage

This function returns any unused characters left over from the street address after it has been submitted.

Remarks

This function may return up to a 50-character string value containing any garbage characters left from the street address after a successful call to the **VerifyAddress** function.

The return value may contain any characters or tokens that were not used by the **VerifyAddress** function to validate the address. Common items found returned by this function are community center names, names of individuals, additional delivery instructions, and so on.

Some examples would be “West Wing” or “Beaumont Main Office.”

Syntax

```
StringValue = object->getParsedGarbage();
```

C

```
StringValue = mdAddrGetParsedGarbage(object);
```

COM

```
StringValue = object.ParsedGarbage
```

2.3.11: Retrieve Demographic and Other Information

The following functions return additional information linked to the submitted address:

GetAddressTypeCode

This function returns the type of address that was coded (PO Box, Rural Route, and so on), using the input address.

Remarks

The **GetAddressTypeCode** function returns a one-character value set by a call to the **VerifyAddress** function.

For U.S. addresses, Address Object would return one of the following codes:

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

For Canadian addresses, Address Object would return one of the following codes:

Code	String
1	Street
2	Street Served By Route and GD
3	Lock Box
4	Route Service
5	General Delivery
B	LVR Street
C	Government Street

Code	String
D	LVR Lock Box
E	Government Lock Box
L	LVR General Delivery
K	Building

If the **VerifyAddress** function has not been called, or if the **VerifyAddress** function call resulted in an error, this function returns an empty value.

This function does not return any data when the GetResults function returns the results code “AS03” (Non-USPS address).

Syntax

```
StringValue = object->GetAddressTypeCode();
```

C

```
StringValue = mdAddrGetAddressTypeCode(object);
```

COM

```
StringValue = object.AddressTypeCode
```

GetAddressTypeString

This function returns a string value that describes the code returned by the **GetAddressTypeCode** function.

Remarks

The **GetAddressTypeString** function returns a string value after a successful call to the **VerifyAddress** function.

For a list of the strings returned by this function, see the section above for the **GetAddressTypeCode** function.

If the **VerifyAddress** function has not been called, or if the **VerifyAddress** function call resulted in an error, this function will return an empty value.

This function does not return any data when the **GetResults** function returns the results code “AS03” (Non-USPS address).

Syntax

```
StringValue = object->GetAddressTypeString();
```

C

```
StringValue = mdAddrGetAddressTypeString(object);
```

COM

```
StringValue = object.AddressTypeString
```

GetCityAbbreviation

This function returns the official 13-letter city name abbreviation associated with the input address.

Remarks

If the value returned by the **GetCity** function is longer than 13 letters, the **GetCityAbbreviation** function will return the official abbreviation the Post Office has associated with that city or municipality name. For example, the **GetCityAbbreviation** function will return the abbreviation “Rcho Sta Marg” for “Rancho Santa Margarita.”

If the value returned by the **GetCity** function is 13 letters or shorter, the **GetCityAbbreviation** function will return the full city or municipality name.

Syntax

```
StringValue = object->GetCityAbbreviation();
```

C

```
StringValue = mdAddrGetCityAbbreviation(object);
```

COM

```
StringValue = object.CityAbbreviation
```

GetCMRA

U.S. Only — This function returns a one-character string that indicates whether or not an address is actually a private mailbox at a Commercial Mail Receiving Agency (CMRA) like the UPS Store. Returned by a successful call to the **VerifyAddress** function.

Remarks

This function returns a one-character value after a successful call to the **VerifyAddress** function:

Code	Description
N	The address does not belong to a CMRA.
Y	The address belongs to a CMRA.
Empty	This field is not populated when the **GetDPVFootnotes** function returns an "F1," "G1," or "U1" code.

This function does not return any data when the **GetResults** function returns the results code "AS03" (Non-USPS address).

Syntax

```
StringValue = object->GetCMRA();
```

C

```
StringValue = mdAddrGetCMRA(object);
```

COM

```
StringValue = object.CMRA
```

GetCongressionalDistrict

U.S. Only — This function returns the congressional district number associated with the input address.

Remarks

The GetCongressionalDistrict function returns a four-character string value after a call to the **VerifyAddress** function. This number is the congressional district for the address given and is accurate to the ZIP + 4 level.

If the **VerifyAddress** function has not been called, or if the **VerifyAddress** function call resulted in an error, this function will return an empty value.

This function returns the standard abbreviation of the state name plus the two-digit number representing the congressional district number. For example, an address in the first district in California would return “CA01.”

For states with only one congressional representative, the value “01” is returned. To find the representative associated with this district, use the Congress.DBF file located on the DVD-ROM in the \data directory. The district number and state abbreviation will be needed to locate this person.

Syntax

```
StringValue = object->GetCongressionalDistrict();
```

C

```
StringValue = mdAddrGetCongressionalDistrict();
```

COM

```
StringValue = object.CongressionalDistrict
```

GetCountyFips

U.S. Only — This function returns the five-digit county FIPS code associated with the input address.

Remarks

The **GetCountyFips** function returns a five-character value string after a successful call to the **VerifyAddress** function.

The Federal Information Processing Standard (FIPS) is a five-digit code defined by the U.S. Bureau of Census. The first two digits are the state code and the last three indicate the county within the state.

For Example: “06037” is the County FIPS for Los Angeles, CA (“06” is the state code for California and “037” is the county code for Los Angeles).

The value returned by the **GetCountyFips** function is accurate to the nine-digit ZIP + 4 level.

Syntax

```
StringValue = object->GetCountyFips();
```

C

```
StringValue = mdAddrGetCountyFips(object);
```

COM

```
StringValue = object.CountyFips
```

GetCountyName

This function returns the name of the county associated with the input address.

Remarks

The GetCountyName function returns a 25-character string value up to 25 characters after a successful call to the **VerifyAddress** function.

The county name is associated with the County FIPS code and it is accurate to the nine-digit ZIP + 4 level.

Syntax

```
StringValue = object->GetCountyName();
```

C

```
StringValue = mdAddrGetCountyName(object);
```

COM

```
StringValue = object.CountyName
```

GetMsa (Deprecated)

U.S. Only — This function returns the MSA number associated with the input address.

Because PMSA and MSA information is no longer updated, this function has been deprecated.

Remarks

The **GetMsa** function returns a four-digit value set by a call to the **VerifyAddress** function.

If the **VerifyAddress** function has not been called, or if the **VerifyAddress** function call resulted in an error, this function will return an empty value.

The Office of Management and Budget defines the Metropolitan Statistical Area (MSA) as one or more counties forming a large population with adjacent communities and having a high degree of social and economic integration.

Syntax

```
StringValue = object->GetMsa();
```

C

```
StringValue = mdAddrGetMsa(object);
```

COM

```
StringValue = object.Msa
```

GetPmsa (Deprecated)

U.S. Only — This function returns the PMSA number associated with the input address.

Because PMSA and MSA information is no longer updated, **this function has been deprecated.**

Remarks

The GetPmsa function returns a four-digit string value set after a successful call to the **VerifyAddress** function.

If the **VerifyAddress** function has not been called, or if the **VerifyAddress** function call resulted in an error, this function will return an empty value.

The Office of Management and Budget defines the Primary Metropolitan Statistical Area (PMSA) for regions that contain a population of more than one million.

Syntax

```
StringValue = object->GetPmsa();
```

C

```
StringValue = mdAddrGetPmsa(object);
```

COM

```
StringValue = object.Pmsa
```

GetPrivateMailbox

U.S. Only — This function returns the private mail box number associated with a CMRA (Commercial Mail Receiving Agency).

Remarks

This function returns a string value after a successful call to the **VerifyAddress** function.

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.

Syntax

```
StringValue = object->GetPrivateMailbox();
```

C

```
StringValue = mdAddrGetPrivateMailbox(object);
```

COM

```
StringValue = object.PrivateMailbox
```

GetTimeZone

This function returns a string value describing the time zone of the input address.

Remarks

All Melissa Data products express time zones in UTC (Coordinated Universal Time).

The **GetTimeZone** function returns a string value describing the time zone of the ZIP where the address is located. If an address cannot be verified, this string will be empty.

These are the strings that can be returned:

“Military”	“Mountain Time”	“Marshall Islands Time”
"Newfoundland Time"	“Pacific Time”	“Guam Time”
“Atlantic Time”	“Alaska Time”	“Palau Time”
“Eastern Time”	“Hawaii Time”	
“Central Time”	“Samoa Time”	

The **GetTimeZone** function does not account for daylight savings time regardless of whether it affects an area or not.

Syntax

```
StringValue = object->GetTimeZone()
```

C

```
StringValue = mdAddrGetTimeZone(object)
```

COM

```
StringValue = object.TimeZone
```

GetTimeZoneCode

This function returns a one or two-digit code representing the time zone associated with the input address.

Remarks

The **GetTimeZoneCode** function returns a one or two-digit number representing the time zone for the address returned by a call to the **VerifyAddress** function. If an address cannot be verified, this string will be empty.

These are the possible values returned by the **GetTimeZoneCode** function:

Code	Zone	Code	Zone
0	Military (APO or FPO)	9	Alaska Time
3A	Newfoundland Time (1/2 Hour Change)	10	Hawaii Time
4	Atlantic Time	11	Samoa Time
5	Eastern Time	12	Marshall Islands Time
6	Central Time	14	Guam Time
7	Mountain Time	15	Palau Time
8	Pacific Time		

The **GetTimeZoneCode** function does not account for daylight savings time.

Syntax

```
StringValue = object->GetTimeZoneCode()
```

C

```
StringValue = mdAddrGetTimeZoneCode(object)
```

COM

```
StringValue = object.TimeZoneCode
```

GetZipType

U.S. Only — This function returns a string value representing the ZIP Code type associated with the submitted address.

Remarks

This function returns a one-character string set by a successful call to the **VerifyAddress** function.

The value returned by **GetZipType** defines the type of ZIP Code for delivery purposes.

Code	Explanation
P	A ZIP Code used only for PO Boxes.
U	Unique: A ZIP Code assigned to an organization or government institution such as the IRS.
M	Military: A ZIP Code assigned to an APO/FPO.

Code	Explanation
Empty	A standard ZIP Code.

Syntax

```
StringValue = object->GetZipType();
```

C

```
StringValue = mdAddrGetZipType(object);
```

COM

```
StringValue = object.ZipType
```

2.3.12: Retrieve Information for Mailing and Sorting

The following functions return information used for presorting lists and printing address labels:

GetCarrierRoute

U.S. Only — This function returns the carrier route associated with the input address.

Remarks

This function returns a four-character string value after a successful call to the **VerifyAddress** function.

The first character of this Carrier Route 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.

B = PO Box

C = City Delivery

G = General Delivery

H = Highway Contract

R = Rural Route

This function does not return any data when the **GetResults** function returns the results code “AS03” (Non-USPS address).

Syntax

```
StringValue = object->GetCarrierRoute();
```

C

```
StringValue = mdAddrGetCarrierRoute(object);
```

COM

```
StringValue = object.CarrierRoute
```

GetDeliveryPointCode

U.S. Only — This function returns the two-digit delivery point code associated with the input address.

Remarks

This function returns a two-digit character string set after a successful return from the **VerifyAddress** function. This two-digit string makes up the 10th and 11th positions of a 12-digit POSTNET™ barcode.

See the explanation for the **GetDeliveryPointCheckDigit** function for a complete guide to producing POSTNET barcodes.

Syntax

```
StringValue = object->GetDeliveryPointCode();
```

C

```
StringValue = mdAddrGetDeliveryPointCode(object);
```

COM

```
StringValue = object.DeliveryPointCode
```

GetDeliveryPointCheckDigit

U.S. Only — This function returns the one-digit number representing the check digit associated with the input address.

Remarks

This function returns a one-digit string set after a successful return from the **VerifyAddress** function. This one-digit string makes up the 12th position of a 12-digit POSTNET barcode.

In 12-digit POSTNET barcodes, the ZIP Code is used for positions 1 to 5, the Plus4 code for positions 6 to 9, the delivery point code for positions 10 and 11, and this check digit for position 12.

Syntax

```
StringValue = object->GetDeliveryPointCheckDigit();
```

C

```
StringValue = mdAddrGetDeliveryPointCheckDigit(object);
```

COM

```
StringValue = object.DeliveryPointCheckDigit
```

GetELotNumber

U.S. Only — This function returns a string containing the Line-of-Travel number for the submitted address.

Remarks

The Line-of-Travel number designates approximately where the submitted address falls within the ZIP + 4.

This number may be required during presorting Standard Mail® when using LOT or Carrier Route sortations.

If the eLOT® data file is not present in the path specified with the **SetPathToDPVDataFiles** function, this function will return a value of "-1."

To get the most value from the return value of this function, use Melissa Data's Presort Object to presort your mailing list.

This function does not return any data when the **GetResults** function returns the results code "AS03" (Non-USPS address).

Syntax

```
StringValue = object->GetELotNumber();
```

C

```
StringValue = mdAddrGetELotNumber(object);
```

COM

```
StringValue = object.ELotNumber
```

GetELotOrder

U.S. Only — This function returns a string containing the Line-of-Travel direction indicator for the submitted address.

Remarks

The Line-of-Travel Order designates whether the Post Office delivers mail in the current ZIP + 4 in ascending or descending order according to the LOT Number.

This function returns an “A” for “ascending” or “D” for “descending.”

This function may be required during presorting Standard Mail when using LOT or Carrier Route sortations.

If the eLOT data file is not present in the path specified with the **SetPathToDPVDataFiles** function, this function will return a value of “X.”

This function does not return any data when the **GetResults** function returns the results code “AS03” (Non-USPS address).

Syntax

```
StringValue = object->GetELotOrder();
```

C

```
StringValue = mdAddrGetELotOrder(object);
```

COM

```
StringValue = object.ELotOrder
```

GetAddressKey

This function return a unique 11-digit address key for the submitted address.

Remarks

The return value of the **GetResults** function is a string value that provides a unique identifier for a given address. This key will be required to use some Melissa Data products with non-USPS addresses.

Syntax

```
StringValue = object->GetAddressKey();
```

C

```
StringValue = mdAddrGetAddressKey(object);
```

COM

```
StringValue = object.AddressKey
```

2.3.13: Suggestions

The following functions return possible correct addresses if the **VerifyAddress** function was unable to verify or correct the submitted address:

FindSuggestion

This function returns the first suggestion based on the original submitted address.

Remarks

This function can be called after the **VerifyAddress** function, if the originally submitted address could not be verified by Address Object.

FindSuggestion will only work if the **SetCASSEnable** function is set to false. You cannot combine suggestions with CASS verification. You must also have DPV enabled, meaning that a valid directory path to the DPV data files must be set using the **SetPathToDPVDataFiles** function.

This function returns an integer value. If the return value is 1, then a suggestion was found and this address was used to populate the same return values as the **VerifyAddress** function. Use the same functions as you would have with **VerifyAddress** (listed above starting on page 81).

A return value of 0 indicates that no suggestion was found.

Syntax

```
BooleanValue = object->FindSuggestion();
```

C

```
IntegerValue = mdAddrFindSuggestion(object);
```

COM

```
IntegerValue = object.FindSuggestion()
```

FindSuggestionNext

This function returns the next suggestion, if any, based on the originally submitted address.

Remarks

The **FindSuggestionNext** function can be called after a successful call to the **FindSuggestion** function and can be repeated until the function returns an integer value of 0, indicating that no further suggestions are available.

If this function returns an integer value of 1, then the suggested address was used to populate the return values of the same functions as the **FindSuggestion** function.

Syntax

```
BooleanValue = object->FindSuggestionNext();
```

C

```
IntegerValue = mdAddrFindSuggestionNext(object);
```

COM

```
IntegerValue = object.FindSuggestionNext()
```

2.3.14: Retrieve CASS Form 3553

U.S. Only — The following functions retrieve CASS Form 3553 and return the information used to populate the form. Address Object only generates a CASS Form 3553 for U.S. addresses. This information is already included on the generated CASS Form, so it is returned here only for information purposes:

GetFormPS3553

The functionality for the **GetFormPS3553** function is the same as GetCASSForm. The GetCASSForm function is found only in the COM object.

This function returns the CASS Form 3553 as a string value in HTML format which can be saved to a file, opened in a Web browser, or put into a Web browser control for viewing and printing. To save the CASS Form directly to an HTML file, use the **SaveFormPS3553** function.

Remarks

CASS (USPS Certification Process for ZIP + 4 Matching Software) improves the accuracy of address matching by providing a common system to measure the quality of address matching software.

CASS Form 3553 verifies to the Post Office that your data has been coded using CASS Certified software, which is a requirement for claiming many discounted postage rates. Form 3553 must accompany a mailing to the Post Office to receive these discounts.

*If the value passed to the **SetCASSEnable** function is FALSE, this function will generate an HTML containing the following message:*

*The CASS 3553 form is currently not available. You must process your mailing with the **SetCASSEnable** flag set to "true."*

Syntax

```
void = object->GetFormPS3553();
```

C

```
void = GetFormPS3553(object);
```

COM

```
object.GetCASSForm()
```

SaveFormPS3553

The functionality for the **SaveFormPS3553** function is the same as SaveCASSForm. The SaveCASSForm function is found only in the COM object.

Saves the CASS Form 3553 as an HTML or PDF file that can be opened in a Web browser or a Web browser control for viewing and printing.

Input Parameters

The **SaveFormPS3553** function has one parameter:

PathAndFileName	The filename, including the path, to which you want to save the CASS Form. You must include the .HTML or .PDF extension in the filename.

Return Value

The **SaveFormPS3553** function returns a Boolean value of 0 (FALSE) if the file cannot be saved, or 1 (TRUE) if the file is saved.

*If the value passed to the **SetCASSEnable** function is FALSE, this function will generate HTML containing the following message:*

*"The CASS 3553 form is currently not available. You must process your mailing with the **SetCASSEnable** flag set to 'true.'"*

Syntax

```
BooleanValue = object->SaveFormPS3553(PathAndFilename);
```

C

```
IntegerValue = mdAddrSaveFormPS3553(object, PathAndFilename);
```

COM

```
BooleanValue = object.SaveCASSForm(PathAndFilename)
```

ResetFormPS3553

The functionality for the **ResetFormPS3553** function is the same as ResetCASSForm. The ResetCASSForm function is found only in the COM object.

Resets all the internal counters to 0 in case you need to produce several CASS forms in sequence for multiple jobs. If you want relevant results on your CASS forms, avoid removing the Address Object from memory as this will clear all the counters.

Syntax

```
void = object->ResetFormPS3553();
```

C

```
void = mdAddrResetFormPS3553(object);
```

COM

```
object.ResetCASSForm()
```

GetPS3553_B6_TotalRecords

This function returns an integer containing the total number of address records verified since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to generate item B6 on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_B6_TotalRecords();
```

C

```
IntegerValue = mdAddrGetPS3553_B6_TotalRecords(object);
```

COM

```
IntegerValue = object.PS3553_B6_TotalRecords
```

GetPS3553_C1a_4Coded

This function returns an integer containing the number of address records which were matched to the ZIP + 4 level since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate item C1a on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_C1a_4Coded();
```

C

```
IntegerValue = mdAddrGetPS3553_C1a_4Coded(object);
```

COM

```
IntegerValue = object.PS3553_C1a_4Coded
```

GetPS3553_C1c_DPBCAssigned (Deprecated)

This function has been deprecated. This field has been removed from the current CASS form and the function will now only return 0. However, this function will not be removed for backward compatibility.

Remarks

Deprecated.

Syntax

```
IntegerValue = object->GetPS3553_C1c_DPBCAssigned();
```

C

```
IntegerValue = mdAddrGetPS3553_C1c_DPBCAssigned(object);
```

COM

```
IntegerValue = object.PS3553_C1c_DPBCAssigned
```

GetPS3553_C1d_FiveDigitCoded

This function returns an integer containing the number of address records which were verified to the five-digit carrier route level since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate item C1d on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_C1d_FiveDigitCoded();
```

C

```
IntegerValue = mdAddrGetPS3553_C1d_FiveDigitCoded(object);
```

COM

```
IntegerValue = object.PS3553_C1d_FiveDigitCoded
```

GetPS3553_C1e_CRRTCoded

This function returns the number of address records which were verified to the carrier route level since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate item C1e on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_C1e_CRRTCoded();
```

C

```
IntegerValue = mdAddrGetPS3553_C1e_CRRTCoded(object);
```

COM

```
IntegerValue = object.PS3553_C1e_CRRTCoded
```

GetPS3553_C1f_eLOTAssigned

This function returns an integer the number of address records which were assigned an enhanced Line of Travel sequence number since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate item C1f on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_C1f_eLOTAssigned();
```

C

```
IntegerValue = mdAddrGetPS3553_C1f_eLOTAssigned(object);
```

COM

```
IntegerValue = object.PS3553_C1f_eLOTAssigned
```

GetPS3553_E_DPVCount (Deprecated)

This function has been deprecated. This field has been removed from the current CASS form and the function will now only return 0. However, this function will not be removed for backward compatibility.

Remarks

Deprecated.

Syntax

```
IntegerValue = object->GetPS3553_E_DPVCount();
```

C

```
IntegerValue = mdAddrGetPS3553_E_DPVCount(object);
```

COM

```
IntegerValue = object.PS3553_E_DPVCount
```

GetPS3553_E_EWSCount

This function returns an integer containing the number of address records that did not appear in the current U. S. Postal Service ZIP + 4 file but were valid addresses since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate section E on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_E_EWSCount();
```

C

```
IntegerValue = mdAddrGetPS3553_E_EWSCount(object);
```

COM

```
IntegerValue = object.PS3553_E_EWSCount
```

GetPS3553_E_HighRiseDefault

This function returns an integer containing the number of high-rise address records that were default matched since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Default matching occurs when the address is successfully matched at the street address level but cannot be matched to a valid secondary address.

Used to populate section E on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_E_HighRiseDefault();
```

C

```
IntegerValue = mdAddrGetPS3553_E_HighRiseDefault(object);
```

COM

```
IntegerValue = object.PS3553_E_HighRiseDefault
```

GetPS3553_E_HighRiseExact

This function returns an integer containing the number of high-rise address records that were exactly matched to both a primary and secondary address since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate section E on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_E_HighRiseExact();
```

C

```
IntegerValue = mdAddrGetPS3553_E_HighRiseExact(object);
```

COM

```
IntegerValue = object.PS3553_E_HighRiseExact
```

GetPS3553_E_LACSCount

This function returns an integer containing the number of address records that were flagged as having undergone a conversion from rural to city-style addresses since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate section E on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_E_LACSCount();
```

C

```
IntegerValue = mdAddrGetPS3553_E_LACSCount(object);
```

COM

```
IntegerValue = object.PS3553_E_LACSCount
```

GetPS3553_E_RuralRouteDefault

This function returns an integer containing the number of rural route address records which were default matched since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Default matches are verified at the route number level but not matched to a valid primary range.

Used to populate section E on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_E_RuralRouteDefault();
```

C

```
IntegerValue = mdAddrGetPS3553_E_RuralRouteDefault(object);
```

COM

```
IntegerValue = object.PS3553_E_RuralRouteDefault
```

GetPS3553_E_RuralRouteExact

This function returns an integer containing the number of rural route address records that were verified at the route number level and also matched to a valid primary range since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate section E on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_E_RuralRouteExact();
```

C

```
IntegerValue = mdAddrGetPS3553_E_RuralRouteExact(object);
```

COM

```
IntegerValue = object.PS3553_E_RuralRouteExact
```

GetPS3553_X_SuiteLinkCodeACount

This function returns an integer containing the number of address records with appended suite information tied to the company name (Suite Link) since the creation of the current instance of Address Object or the most recent call to the **ResetFormPS3553** function.

Remarks

Used to populate section E on CASS Form 3553.

Syntax

```
IntegerValue = object->GetPS3553_X_SuiteLinkCodeACount();
```

C

```
IntegerValue = mdAddrGetPS3553_X_SuiteLinkCodeACount(object);
```

COM

```
IntegerValue = object.PS3553_X_SuiteLinkCodeACount
```

2.3.15: Retrieve Summary of Addresses

Canada Only — The following functions retrieve the Canada Post Summary of Addresses (SOA) and return data used to populate the form. These functions only apply when using the Canadian add-on for verifying Canadian addresses:

GetFormSOA

This function retrieves a string value containing the current Summary of Addresses in HTML format.

Remarks

This function retrieves the contents of the current Summary of Addresses (SOA) for all records processed since Address Object was initialized or the last time the **ResetFormSOA** function was called. The **SetPathToCanadaFiles** function must be set before processing Canadian addresses. You must also call the **SetSOACustomerInfo** and **SetSOACPCNumber** functions before calling this function.

The SOA is good for one year after it is generated. You do not need to generate another form unless your address data changes.

The SOA shows the following information:

Customer Name
Company name as it appears on your Canada Post contract. This is entered using the **SetSOACustomerInfo** function.

Customer Address
Your mailing address as it appears on your Canada Post contract.This is entered using the **SetSOACustomerInfo** function.

Total Records
Total number of Canadian address records processed. This value is also returned by the **GetSOATotalRecords** function.

Address Accuracy
Percentage of Accurate Canadian address records, calculated to one decimal place. This must be 95% or postage discounts will be adjusted. This value is also returned by the **GetSOAAAPercentage** function.

SOA Expiry Date
The Expiry Date for the current SOA. This date will be one year after the SOA is generated. This value is also returned by the **GetSOAAAExpiryDate** function.

Software Information

The company that produced the software used to generate the SOA (Melissa Data Corp.) plus the version number of the software used (the Build Number as returned by the **GetBuildNumber** function (See *GetBuildNumber* on page 28).

CPC Database Date

The date of the CPC database used to verify the accuracy of the database. This is the same date returned by the **GetCanadianDatabaseDate** function.(See *GetCanadianDatabaseDate* on page 29).

Return Value

This function returns a string value containing the HTML code for displaying the SOA in a Web browser.

Syntax

```
StringValue = object->GetFormSOA();
```

C

```
StringValue = mdAddrGetFormSOA(object);
```

COM

```
StringValue = object.GetFormSOA()
```

SaveFormSOA

This function saves an HTML file containing the current Summary of Addresses.

Remarks

This function retrieves and stores the contents of the current Summary of Addresses (SOA) for all records processed since Address Object was initialized or the last time the **ResetFormSOA** function was called. The **SetPathToCanadaFiles** function must be called before processing Canadian addresses. You must also call the **SetSOACustomerInfo** and **SetSOACPCNumber** functions before calling this function.

The SOA is good for one year after it is generated. You do not need to generate another form unless your address data changes.

For a detailed explanation of the SOA report, see *GetFormSOA* on page 123.

Input Parameters

The **SaveFormSOA** function has one parameter:

PathAndFileName	The filename, including the path, to which you want to save the SOA. You must include the .HTML extension in the filename.

Syntax

```
void = object->SaveFormSOA(PathAndFileName);
```

C

```
void = mdAddrSaveFormSOA(object, PathAndFileName);
```

COM

```
object.SaveFormSOA(PathAndFileName)
```

ResetFormSOA

Resets the total records and Address Accuracy percentage for Canadian address data.

Remarks

Use this function to reset Address Object's running totals for Canadian addresses. This function must be called if you want to process multiple databases with the same instance of Address Object.

It is good practice to always call this function before processing Canadian address.

Syntax

```
void = object->ResetFormSOA();
```

C

```
void = mdAddrResetFormSOA(object);
```

COM

```
object.ResetFormSOA
```

GetSOAAAExpiryDate

This function returns a string value containing the expiry date of the current Summary of Addresses.

Remarks

This number also appears on the Summary of Addresses.

Syntax

```
StringValue = object->GetSOAAAExpiryDate();
```

C

```
StringValue = mdAddrGetSOAAAExpiryDate(object);
```

COM

```
StringValue = object.SOAAAExpiryDate
```

GetSOAAAPercentage

This function returns a single-precision value containing the percentage of accurate Canadian records processed since the **ResetFormSOA** function was called or the current instance of Address Object was created.

Remarks

This number also appears on the Summary of Addresses. The Address Accuracy must be at least 95% to avoid adjustments to the postage discounts.

Syntax

```
StringValue = object->GetSOAAAPercentage();
```

C

```
StringValue = mdAddrGetSOAAAPercentage(object);
```

COM

```
StringValue = object.SOAAAPercentage
```

GetSOAErrorString

This function returns a string value containing error information generated by a call to any Summary of Addresses functions.

Remarks

This function will usually only return a value if an SOA function is called without first passing a valid path to the **SetPathToCanadaFiles** function.

Syntax

```
StringValue = object->GetSOAErrorString();
```

C

```
StringValue = mdAddrGetSOAErrorString(object);
```

COM

```
StringValue = object.SOAErrorString
```

GetSOASoftwareInfo

This function returns a string value containing the software information that appears on the current Summary of Addresses.

Remarks

This information also appears on the Summary of Addresses. It contains the creator of the software being used (Melissa Data Corp.) and the version number (build number).

Syntax

```
StringValue = object->GetSOASoftwareInfo();
```

C

```
StringValue = mdAddrGetSOASoftwareInfo(object);
```

COM

```
StringValue = object.SOASoftwareInfo
```

GetSOATotalRecords

This function returns a long integer value containing the total number of Canadian Records processed since the **ResetFormSOA** function was called or the current instance of Address Object was created.

Remarks

This number also appears on the Summary of Addresses.

Syntax

```
StringValue = object->GetSOATotalRecords();
```

C

```
StringValue = mdAddrGetSOATotalRecords(object);
```

COM

```
StringValue = object.SOATotalRecords
```

2.3.16: Melissa Address Key

The Melissa Address Key (MAK) is a new output available only from Melissa data.

The MAK is a time persistent key that identifies an address with a unique number. This number will stay attached to that address and will not change even if parts of that address change. This is what sets the MAK apart from our current AddressKey output. With AddressKey, if an address changes zip code, zip+4, or gets a new delivery point, their AddressKey would change. Under MAK, it will not. So, MAK is a good way to permanently identify and locate addresses. With this new system, there are two new outputs available:

MAK	The Melissa Address Key number for this address.
BASEMAK	This will give you the unique MAK number for the building, if the MAK address is a suite or apartment address.

SetPathToAddrKeyDataFiles

This function sets the path for the Melissa Address Key (MAK) files.

Syntax

```
object->SetPathToAddrKeyDataFiles(StringValue);
```

C

```
mdAddrSetPathToAddrKeyDataFiles(object, char*);
```

COM

```
object.PathToAddrKeyDataFiles = StringValue
```

GetMelissaAddressKey

This function returns the Melissa Address Key (MAK), a proprietary unique key identifier for an address.

Syntax

```
StringValue = object->GetMelissaAddressKey();
```

C

```
StringValue = mdAddrGetMelissaAddressKey(object);
```

COM

```
StringValue = object.MelissaAddressKey
```

GetMelissaAddressKey Base

This function returns the Melissa Address Key Base (BASEMAK), a proprietary unique key identifier for sub-premise addresses.

Syntax

```
StringValue = object->GetMelissaAddressKeyBase();
```

C

```
StringValue = mdAddrGetMelissaAddressKeyBase(object);
```

COM

```
StringValue = object.MelissaAddressKeyBase
```

CHAPTER 3
StreetData Interface

The StreetData Interface is provided as a means to access U.S. street address data more directly than through the AddressCheck Interface.

Reasons to access this data may include showing customers alternate addresses if the address checking logic is unable to code their address. All records that match their range can be displayed, allowing the correction of misspellings. Other uses include filtering our data to meet your specific requirements, as you may only need the street names in a particular carrier route, congressional district, or county FIPS code.

The records returned by the StreetData Interface contain address ranges, but that does not mean that each address in that range is a deliverable address. For example, you may see a range of 1 to 10 Main Street. The deliverable addresses may be 1, 2, 3, 5, 6, 9, and 10. Notice that 4, 7, and 8 will not be deliverable. These records are

effective for checking existing addresses, but not for creating addresses, which is prohibited by the USPS.

3.1: Using the StreetData Interface

1 Create an instance of the Address Object's StreetData Interface.

```
Set streetPtr in New Instance of StreetData
```

2 Connect the StreetData Interface to its data files by calling the **Initialize** function, passing the paths to the files `mdAddr.dat`, and `mdAddr.nat`.

```
CALL Initialize WITH DataPaths RETURNING Result
IF Result <> 0 Then
   Call GetInitializeErrorString RETURNING ErrorString
   PRINT ErrorString
   End If
```

3 Call the **FindStreet** function, passing the street name or wildcard pattern that you want to match.

```
CALL FindStreet WITH StreetName, Code RETURNING Result
IF Result is TRUE THEN
   Call GetAddressType RETURNING AddressType
   CALL GetPrimaryRangeLow RETURNING RangeLow
   CALL GetPrimaryRangeHigh RETURNING RangeHigh
   CALL GetPrimaryRangeOddEven RETURNING RangeOddEven
   CALL GetPreDirection RETURNING PreDirection
   CALL GetStreetName RETURNING StreetName
   CALL GetSuffix RETURNING Suffix
   CALL GetPostDirection RETURNING PostDirection
   CALL GetSuiteName RETURNING SuiteName
   CALL GetSuiteRangeHigh RETURNING SuiteHigh
   CALL GetSuiteRangeLow RETURNING SuiteLow
   CALL GetSuiteRangeOddEven RETURNING SuiteOddEven
   CALL GetZip RETURNING Zip
   CALL GetPlus4Low RETURNING Plus4Low
   CALL GetPlus4High RETURNING Plus4High

   WRITE VALUES TO LIST OR SPREADSHEET

End If
```

4 Continue to call the **FindStreetNext** function until it returns a FALSE value to retrieve the remaining matches for the pattern passed to the **FindStreet** function.

```
CALL FindStreetNext RETURNING Result
WHILE Result Is TRUE
    Call GetAddressType RETURNING AddressType
    CALL GetPrimaryRangeLow RETURNING RangeLow
    CALL GetPrimaryRangeHigh RETURNING RangeHigh
    CALL GetPrimaryRangeOddEven RETURNING RangeOddEven
    CALL GetPreDirection RETURNING PreDirection
    CALL GetStreetName RETURNING StreetName
    CALL GetSuffix RETURNING Suffix
    CALL GetPostDirection RETURNING PostDirection
    CALL GetSuiteName RETURNING SuiteName
    CALL GetSuiteRangeHigh RETURNING SuiteHigh
    CALL GetSuiteRangeLow RETURNING SuiteLow
    CALL GetSuiteRangeOddEven RETURNING SuiteOddEven
    CALL GetZip RETURNING Zip
    CALL GetPlus4Low RETURNING Plus4Low
    CALL GetPlus4High RETURNING Plus4High

    WRITE Values TO Range Lost

    CALL FindStreetNext RETURNING Result
ENDWHILE
```

In reality, you would probably use the **FindStreet** and **FindStreetNext** functions to populate a list, array, or collection.

5 After retrieving a list of matching address ranges, you can loop through the list results and compare a parsed address to the ranges in each result. The street number would be from an instance of the Parse Interface.

```
FOR EACH Range IN LIST
    IF IsAddressInRange2(StreetNum,RangeHigh,RangeLow,
           RangeOddEven) THEN
       Match Is TRUE
    ENDIF
NEXT
```

3.2: StreetData Functions

3.2.1: Initialize the StreetData Interface

Initializing the StreetData Interface is slightly different than the procedure for the AddressCheck Interface.

3.2.2: Search for Street Addresses

The FindStreet function locates the first range of street addresses that matches a submitted pattern and ZIP Code.

3.2.3: Retrieve the Street Range Data

The following functions retrieve the data from one range of street addresses that match a pattern submitted via the FindStreet function. An application can then cycle through multiple results using the FindStreetNext function.

3.2.4: Retrieve the Next Record

The FindStreetNext function resets the return values of the above functions with the next matching street data record.

3.2.5: Check Address Against Results

This function allows you to check a parsed address against the ranges returned by the FindStreet function.

3.2.6: Auto-Complete

This function allows you to enable auto-complete for StreetData in Address Object.

3.2.1: Initialize the StreetData Interface

Initializing the StreetData Interface is slightly different than the procedure for the AddressCheck Interface.

SetLicenseString

This function sets the license string required to enable Address Object's complete functionality.

Remarks

The License String is a software key that unlocks the full functionality of the component. Without the License String, the object will only function in DEMO mode (Nevada only).

The license string is normally set using an environment variable, either MD_LICENSE or MD_LICENSE_DEMO. Calling SetLicenseString is an alternative method for setting the license string, but applications developed for a production environment should only use the environment variable.

When using an environment variable, it is not necessary to call the **SetLicenseString** function.

For more information on setting the environment variable, see page 3 of this guide.

Input Parameters

The **SetLicenseString** function has one parameter:

LicenseString	A string value representing the software license key.

Return Value

The **SetLicenseString** function returns a Boolean value of 0 (FALSE) or 1 (TRUE). It will return a FALSE Boolean value if the License String provided is incorrect.

Syntax

```
BooleanValue = object->SetLicenseString(LicenseString);
```

C

```
IntegerValue = mdStreetSetLicenseString(object, LicenseString);
```

COM

```
BooleanValue = object.SetLicenseString(LicenseString)
```

GetBuildNumber

The **GetBuildNumber** function returns a string value containing the current development release build number of the Address Object.

Remarks

This is usually a three or four-character string. If you are running the Demo version of Address Object, the word DEMO will be appended to the build number.

Input Parameters

None.

Return Value

The **GetBuildNumber** function returns a string value containing the current development release build number of the Address Object.

Syntax

C++

```
StringValue = object->GetBuildNumber();
```

C

```
StringValue = mdStreetGetBuildNumber(object);
```

COM

```
StringValue = object.GetBuildNumber()
```

GetDatabaseDate

The **GetDatabaseDate** function returns the date of your address data files.

Remarks

If the **GetDatabaseDate** function is called before the **Initialize** function is called and the data files are not in the same directory as Address Object, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

Input Parameters

None.

Return Value

The **GetDatabaseDate** function returns a value that represents the date of your address data files. The COM object returns a date value, while the standard object returns a string value.

Syntax

```
StringValue = object->GetDatabaseDate();
```

C

```
StringValue = mdStreetGetDatabaseDate(object);
```

COM

```
DateTime = object.GetDatabaseDate()
```

Initialize

Opens the required data file and prepares the address checking logic for use. If the data file is in the same directory as the Component, you do not need to call the **Initialize** function. However, if there is an error, an exception will be reported instead of an error code.

Remarks

If the **Initialize** function returns a code other than zero, you can call the **GetInitializeErrorString** function to display a string describing the error.

Input Parameters

The **Initialize** function has three string values as parameters. In VB and languages that support optional parameters, all three are considered optional. If the path is not provided, the **Initialize** function will look for the data files in the same directory as the Address Object component file.

DataPath	A string value containing the path to the location of the `mdAddr.dat` file.
NationalPath	A string value containing the path to the location of the `mdAddr.nat` file.
RegionalPath	This parameter is deprecated and no longer used. It is included for backwards compatibility. If your programming language supports optional parameters, you can ignore RegionalPath completely. If your programming language does not allow optional parameters, you must pass something to this parameter, even if it is only an empty string value.

Return Value

The **Initialize** function returns one of the following:

Code	Description
0	No error - initialization was successful.
1	Could not open the `mdAddr.dat` file.
2	Could not open the `mdAddr.nat` or `mdAddr.str` file.
4	The internal database date of the `mdAddr.dat` and `mdAddr.nat` files do not match.
5	Not all the memory buffers could be initialized.
6	Unknown error.

Syntax

```
IntegerValue = object->Initialize(DataPath, NationalPath,
RegionalPath);
```

C

```
IntegerValue = mdStreetInitialize(object, DataPath,
NationalPath, RegionalPath);
```

COM

```
IntegerValue = object.Initialize(DataPath, NationalPath,
RegionalPath)
```

GetInitializeErrorString

This function returns a descriptive string to describe an error from the **Initialize** function.

Remarks

The **GetInitializeErrorString** function returns a string describing the error that occurred when the **Initialize** function failed.

Syntax

```
StringValue = object->GetInitializeErrorString();
```

C

```
StringValue = mdStreetGetInitializeErrorString(object);
```

COM

```
StringValue = object.GetInitializeErrorString()
```

GetLicenseExpirationDate

This function returns a date value corresponding to the date when the current license string expires.

Remarks

License strings issued by Melissa Data are valid a certain period of time. This function returns the date after which the current license string is no longer valid.

The COM object returns a date value, while the standard object returns a string value.

Syntax

```
StringValue = object->GetLicenseExpirationDate();
```

C

```
StringValue = mdStreetGetLicenseExpirationDate(object);
```

COM

```
DateTime = object.GetLicenseExpirationDate
```

3.2.2: Search for Street Addresses

The **FindStreet** function locates the first range of street addresses that matches a submitted pattern and ZIP Code.

FindStreet

This function retrieves the first address range that matches a given street name pattern and ZIP Code.

Input Parameters

The **FindStreet** function requires the following parameters:

StreetName	String value. Wildcard allowed, but only at the end of the string value. For example, “MA*” for any streets beginning with “MA.”
ZIP	String value containing the five-digit ZIP Code.
CodeOnly	Boolean value defaulted to 0. If 1 is passed, FindStreet will return street records only within the supplied ZIP Code. If 0, FindStreet will also return any matching records within the city in which the ZIP Code is located.

Return Value

The **FindStreet** function returns a Boolean value of 0 (FALSE) to indicate no records were found, or 1 (TRUE) to indicate a match was made.

If a match was made, call the following functions to retrieve the street data record:

If the **FindStreet** function is used with a Canadian address, the return values of the **GetPlus4Low**, **GetPlus4High**, **GetCarrierRoute**, **GetCountyFips**, **GetCongressionalDistrict**, **GetLastLineNumber**, **GetUrbanizationCode**, **GetUrbanizationName**, **GetLACSIndicator**, and **GetBaseAlternateIndicator** functions will not be populated.

Syntax

```
BooleanValue = object->FindStreet(StreetName, Zip[,
ZipCodeOnly]);
```

C

```
IntegerValue = mdStreetFindStreet(object StreetName, Zip,
ZipCodeOnly);
```

COM

```
IntegerValue = object.FindStreet(StreetName, Zip[,
ZipCodeOnly])
```

3.2.3: Retrieve the Street Range Data

The following functions retrieve the data from one range of street addresses that match a pattern submitted via the **FindStreet** function. An application can then cycle through multiple results using the **FindStreetNext** function.

GetPrimaryRangeLow

This function returns the lowest street number in each street data record.

Remarks

The **GetPrimaryRangeLow** function returns a 10-character string value after a call to the **FindStreet** or **FindStreetNext**.

The return value gives the first number in the address range. For example, if the address range is 100 to 200 Main Street, this function will return the “0000000100” value.

All numeric data returned is padded with zeroes on the left.

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

No hyphen symbol (-) indicates that the leading zeros are not a part of this range.

Syntax

```
StringValue = object->GetPrimaryRangeLow();
```

C

```
StringValue = mdStreetGetPrimaryRangeLow(object);
```

COM

```
StringValue = object.PrimaryRangeLow
```

GetPrimaryRangeHigh

This function returns the highest street number in each street data record.

Remarks

The **GetPrimaryRangeHigh** function returns a 10-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

The return value contains the first number in the address range. For example, if the address range is 100 to 200 Main Street, this function will return the “0000000200” value.

All numeric data returned is padded with zeroes on the left.

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

No hyphen symbol (-) indicates that the leading zeros are not a part of this range.

Syntax

```
StringValue = object->GetPrimaryRangeHigh();
```

C

```
StringValue = mdStreetGetPrimaryRangeHigh(object);
```

COM

```
StringValue = object.PrimaryRangeHigh
```

GetPrimaryRangeOddEven

This function returns the Odd/Even indicator for the street number range in each street data record.

Remarks

The **GetPrimaryRangeOddEven** function returns a one-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

This one-character value shows either an “O” for Odd, an “E” for Even, or a “B” for Both. An “O” indicates that the address range contains only odd numbered addresses and an “E” indicates that only even numbered addresses are present in the address range. A “B” indicates that both odd and even address numbers are present in the

address range. For example, an “O” means that in the 101 to 201 address range, only the numbers 101, 103, 105, 107, 109, and so on, are valid address numbers.

Syntax

```
StringValue = object->GetPrimaryRangeOddEven();
```

C

```
StringValue = mdStreetGetPrimaryRangeOddEven(object);
```

COM

```
StringValue = object.PrimaryRangeOddEven
```

GetPreDirection

This function returns the geographic directional that precedes the address range returned, if any.

Remarks

The **GetPreDirection** function returns a two-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

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

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

Syntax

```
StringValue = object->GetPreDirection();
```

C

```
StringValue = mdStreetGetPreDirection(object);
```

COM

```
StringValue = object.PreDirection
```

GetStreetName

This function returns the name of the street for each street data record.

Remarks

The GetStreetName function returns a 28-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

For street names that begin with one of the following Spanish words, that first word will be returned by the **GetSuffix** function and the next word in the street name will be returned as the street name. This is because Spanish street names have the suffix at the front of the address. If the suffix was not returned by the **GetSuffix** function, it would appear that every street in some Puerto Rican ZIP codes began with the same letter. The Spanish words that would be returned by the **GetSuffix** function are: “Avenida;” “Calle;” “Camino;” “Paseo;” and “Via.”

Syntax

```
StringValue = object->GetStreetName();
```

C

```
StringValue = mdStreetGetStreetName(object);
```

COM

```
StringValue = object.StreetName
```

GetSuffix

This function returns the street suffix for each street data record.

Remarks

The **GetSuffix** function is a four-character maximum string value set by a call to the **FindStreet** or **FindStreetNext** function.

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

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

Syntax

```
StringValue = object->GetSuffix();
```

C

```
StringValue = mdStreetGetSuffix(object);
```

COM

```
StringValue = object.Suffix
```

GetPostDirection

This function returns the directional that follows the street name (if any) for the address range returned.

Remarks

The **GetPostDirection** function returns a two-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

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

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

Syntax

```
StringValue = object->GetPostDirection();
```

C

```
StringValue = mdStreetGetPostDirection(object);
```

COM

```
StringValue = object.PostDirection
```

GetSuiteName

This function returns the name for the secondary range (if any) of each street record.

Remarks

The **GetSuiteName** function returns a four-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

If suite numbers exist in the address range, the return value of this function will indicate the proper suite name for addresses in that range.

Possible return values are:

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

Syntax

```
StringValue = object->GetSuiteName();
```

C

```
StringValue = mdStreetGetSuiteName(object);
```

COM

```
StringValue = object.SuiteName
```

GetSuiteRangeLow

This function returns the lowest number in the suite range associated with each address record returned.

Remarks

The **GetSuiteRangeLow** function returns an eight-character maximum string value set by a successful call to the **FindStreet** or **FindStreetNext** function.

The return value gives the first number in the suite range. For example, this function will return the "1001" end of a suite range from 1001 to 2001.

Syntax

```
StringValue = object->GetSuiteRangeLow()
```

C

```
StringValue = mdStreetGetSuiteRangeLow(object);
```

COM

```
StringValue = object.SuiteRangeLow
```

GetSuiteRangeHigh

This function returns the highest number in the suite range associated with each street data record.

Remarks

The **GetSuiteRangeHigh** function returns an eight-character maximum string value after a successful call to the **FindStreet** or **FindStreetNext** function.

The return value gives the last number in the suite range. For example, this function will return the "2001" end in the 1001 to 2001 suite range.

Syntax

```
StringValue = object->GetSuiteRangeHigh();
```

C

```
StringValue = mdStreetGetSuiteRangeHigh(object);
```

COM

```
StringValue = object.Suffix
```

GetSuiteRangeOddEven

This function returns the odd/even indicator for the suite range associated with each street data record.

Remarks

The **GetSuiteRangeOddEven** function returns a one-character string value after a call to the **FindStreet** or **FindStreetNext** function.

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

Syntax

```
StringValue = object->GetSuiteRangeOddEven();
```

C

```
StringValue = mdStreetGetSuiteRangeOddEven(object);
```

COM

```
StringValue = object.SuiteRangeOddEven
```

GetZip

This function returns the five-digit ZIP Code for each address record returned.

Remarks

The **GetZip** returns the five-digit ZIP Code for the current record.

Syntax

```
StringValue = object->GetZip();
```

C

```
StringValue = mdStreetGet(object);
```

COM

```
StringValue = object.
```

GetPlus4Low

This function returns the lowest number in the Plus4 range for each address record returned.

Remarks

The **GetPlus4Low** function returns a four-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

This return value gives the first Plus4 add-on in a ZIP + 4 range. For example, if the ZIP + 4 range is 1234 to 1334, this field will show the “1234” value.

The **GetPlus4Low** and **GetPlus4High** function return values will usually contain the same value. In some instances, such as with PO Boxes, the **GetPlus4Low** function and **GetPlus4High** function will return different values as every PO Box has its own Plus4 code.

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

If the Plus4Low value is “xxxx,” this street record is deemed undeliverable by the USPS.

This function does not return a value if **FindStreet** was called with a Canadian address.

Syntax

```
StringValue = object->GetPlus4Low();
```

C

```
StringValue = mdStreetGetPlus4Low(object);
```

COM

```
StringValue = object.Plus4Low
```

GetPlus4High

This function returns the highest number in the Plus4 range for each address record returned.

Remarks

The **GetPlus4High** function returns a four-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

This field gives the last Plus4 add-on for a ZIP + 4 range. For example, if the ZIP + 4 range is 1234-1334, this field will show the “1334” value.

The **GetPlus4Low** and **GetPlus4High** function return values will usually contain the same value. In some instances, such as with PO Boxes, the **GetPlus4Low** and **GetPlus4High** functions will return different values as every PO Box has its own Plus4 code.

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

If the Plus4Low value is “xxxx,” this street record is deemed undeliverable by the USPS.

This function does not return a value if **FindStreet** was called with a Canadian address.

Syntax

```
StringValue = object->GetPlus4High();
```

C

```
StringValue = mdStreetGetPlus4High(object);
```

COM

```
StringValue = object.Plus4High
```

GetAddressType

This function returns the address type for the address record returned after a successful call to the **FindStreet** or **FindStreetNext** function.

Remarks

The **GetAddressType** function returns a 1-character (maximum) string value after a call to the **FindStreet** or **FindStreetNext** function. This 1-character code indicates the type of address that was returned:

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

Syntax

```
StringValue = object->GetAddressType();
```

C

```
StringValue = mdStreetGetAddressType(object);
```

COM

```
StringValue = object.AddressType
```

GetBaseAlternateIndicator

This function returns the base alternate indicator for the address record returned after a successful call to the **FindStreet** or **FindStreetNext** function.

Remarks

This function returns a 1-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

This code specifies whether or not a record is a base (preferred) or alternate record. Base records (indicated by a “B”) can represent a range of addresses or an individual address, such as a firm record, while alternate records (indicated by an “A”) are

individual delivery points. Base records are generally preferred over alternate records. The base record for an alternate record can be found by matching up the ZIP + 4 ranges.

This function does not return a value if **FindStreet** was called with a Canadian address.

Syntax

```
StringValue = object->GetBaseAlternateIndicator();
```

C

```
StringValue = MdStreetGetBaseAlternateIndicator(object);
```

COM

```
StringValue = object.BaseAlternateIndicator
```


GetCarrierRoute

This function returns the carrier route for each address record returned by a successful call to the **FindStreet** or **FindStreetNext** function.

Remarks

This function returns a four-character string value after a successful call to the **FindStreet** or **FindStreetNext** function.

The first character returned by this function 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.

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

This function does not return a value if **FindStreet** was called with a Canadian address.

Syntax

```
StringValue = object->GetCarrierRoute();
```

C

```
StringValue = mdStreetGetCarrierRoute(object);
```

COM

```
StringValue = object.CarrierRoute
```

GetCompany

This function returns the name of a company, building, apartment complex, shopping center, or other secondary name information associated with the record returned following a successful call to the **FindStreet** or **FindStreetNext** function.

Remarks

The **GetCompany** function returns a 40-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function.

The return value will contain the company name for each record returned, if any.

Syntax

```
StringValue = object->GetCompany();
```

C

```
StringValue = mdStreetGetCompany(object);
```

COM

```
StringValue = object.Company
```

GetCongressionalDistrict

This function returns the congressional district associated with the address record returned.

Remarks

The **GetCongressionalDistrict** function returns a four-character (maximum) string value after a call to the **FindStreet** or **FindStreetNext** function containing the standard abbreviation of the state name plus the two-digit number representing the congressional district number. For example, an address in the first district in California would return “CA01.”

For states with only one congressional representative, the value “01” is returned. To find the representative associated with this district, use the Congress.DBF file located on the DVD-ROM in the `\data` directory. The district number and state abbreviation will be needed to locate this person.

This function does not return a value if **FindStreet** was called with a Canadian address.

Syntax

```
StringValue = object->GetCongressionalDistrict();
```

C

```
StringValue = mdStreetGetCongressionalDistrict();
```

COM

```
StringValue = object.CongressionalDistrict
```

GetCountyFips

This function returns the county FIPS code associated with each address record returned by a call to the **FindStreet** or **FindStreetNext** function.

Remarks

The **GetCountyFips** function returns a five-character (maximum) string value after a call to the **FindStreet** or **FindStreetNext** function, containing Federal Information Processing Standard (FIPS), a five-digit code defined by the U.S. Bureau of Census. The first two digits are the state code and the last three indicate the county within the state.

For Example: “06037” is the County FIPS for Los Angeles, CA (“06” is the state code for California and “037” is the county code for Los Angeles).

The return value is accurate to the nine-digit ZIP + 4 level.

This function does not return a value if **FindStreet** was called with a Canadian address.

Syntax

```
StringValue = object->GetCountyFips();
```

C

```
StringValue = mdStreetGetCountyFips(object);
```

COM

```
StringValue = object.CountyFips
```

GetLACSIndicator

This function returns the LACS indicator for each address record that is returned by a successful call to the **FindStreet** or **FindStreetNext** function.

Remarks

The **GetLacsIndicator** function returns a one-character value after a call to the **FindStreet** or **FindStreetNext** function.

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

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

The new addresses are not contained within the ZIP + 4 file. To change the old addresses to the new addresses, you will either need to send these addresses to a company that does LACS^{Link} processing or use the LACS^{Link} functionality available through the AddressCheck Interface.

This function does not return a value if **FindStreet** was called with a Canadian address.

Syntax

```
StringValue = object->GetLACSIndicator();
```

C

```
StringValue = mdStreetGetLACSIndicator(object);
```

COM

```
StringValue = object.LACSIndicator
```


GetLastLineNumber

This function returns the last line number for each address record returned.

Remarks

The **GetLastLineNumber** function returns a six-character (maximum) string value after a call to the **FindStreet** or **FindStreetNext** function.

The last line number is a six-character string used for advanced address matching. This number can be associated with last line numbers returned from the city name in the Data Interface to break ties based off of city names.

This number is used with address matching to break ties on certain records using the city name.

This function does not return a value if **FindStreet** was called with a Canadian address.

Syntax

```
StringValue = object->GetLastLineNumber();
```

C

```
StringValue = mdStreetGetLastLineNumber(object);
```

COM

```
StringValue = object.LastLineNumber
```

GetUrbanizationCode

This function returns the urbanization code for each street data record.

Remarks

The GetUrbanizationCode returns a six-character maximum string value after a call to the **FindStreet** or **FindStreetNext** function. This code is used for addresses in Puerto Rico only.

Syntax

```
StringValue = object->GetUrbanizationCode();
```

C

```
StringValue = mdStreetGetUrbanizationCode(object);
```

COM

```
StringValue = object.SuiteRangeOddEven
```

GetUrbanizationName

This function returns the urbanization name for each street data record.

Remarks

The **GetUrbanizationName** function returns a string value after a call to the **FindStreet** or **FindStreetNext** function. The urbanization name is used for addresses in Puerto Rico only.

Syntax

```
StringValue = object->GetUrbanizationName();
```

C

```
StringValue = mdStreetGetUrbanizationName(object);
```

COM

```
StringValue = object.UrbanizationName
```

GetDeliveryInstallation

Canadian Addresses Only — This function returns the Delivery Installation information from a Canadian street address.

Remarks

The GetDeliveryInstallation function returns a string value after a call to the **FindStreet** or **FindStreetNext** function.

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

```
StringValue = object->GetDeliveryInstallation();
```

C

```
StringValue = mdStreetGetDeliveryInstallation(object);
```

COM

```
StringValue = object.DeliveryInstallation
```

3.2.4: Retrieve the Next Record

The **FindStreetNext** function resets the return values of the above functions with the next matching street data record.

FindStreetNext

This function retrieves the next StreetData record and sets the return values of the StreetData functions, based on the previous call to the **FindStreet** function.

Remarks

If a match was made, call the functions detailed above to access the street data return values.

A TRUE return will allow the developer to recall the next street record data using the above-mentioned functions, after a call has initially been made to the **FindStreet** function.

A FALSE return means that the **FindStreetNext** function was unable to find any more matching street records.

If the **FindStreet** function was used with a Canadian address, the return values of the **GetPlus4Low**, **GetPlus4High**, **GetCarrierRoute**, **GetCountyFips**, **GetCongressionalDistrict**, **GetLastLineNumber**, **GetUrbanizationCode**, **GetUrbanizationName**, **GetLACSIndicator**, and **GetBaseAlternateIndicator** functions will not be populated.

Input Parameters

The **FindStreetNext** function requires that the **FindStreet** function be called first.

Return Value

The **FindStreetNext** function returns a Boolean value of 0 (FALSE) or 1 (TRUE).

Syntax

```
BooleanValue = object->FindStreetNext();
```

C

```
IntegerValue = mdStreetFindStreetNext(object);
```

COM

```
IntegerValue = object.FindStreetNext
```

3.2.5: Check Address Against Results

This function allows you to check a parsed address against the ranges returned by the **FindStreet** function.

IsAddressInRange2

The **IsAddressInRange2** function determines if an address, when compared against a USPS address record, falls within the address range of the USPS address record *and* matches the record at the Odd/Even level.

Remarks

Although this appears to be a simple process, alpha-numeric addresses such as “Five,” “105B” or “19 1/2,” which this function is equipped to handle, should be taken into account.

If the **VerifyAddress** function indicates multiple matches, using this function in conjunction with the **FindStreet** and **FindStreetNext** functions makes it relatively simple to match the submitted address to a valid address record.

If you previously used the **IsAddressInRange** function, this has been superceded by the **IsAddressInRange2** function and it is now deprecated. The new function was introduced because this function did not consider whether the address range to be tested contained odd numbers, even numbers, or both.

Input Parameters

The **IsAddressInRange2** function takes the following four parameters in the order given: *InputRange*; *LowRange*; *HiRange;* and *OddEvenFlag*. The *LowRange*, *HiRange*, and *OddEvenFlag* are obtained with the **GetPrimaryRangeLow**, **GetSuiteRangeHigh**, and **GetPrimaryRangeOddEven** functions after a successful call to the **FindStreet** and **FindStreetNext** functions.

Return Value

The **IsAddressInRange2** function returns a **1** if the input is in range and matches the Odd/Even range. It returns a **0** if the input falls out of range or does not match the Odd/Even range.

Examples

123 Main St. will return a 1 if “123” falls within the range and this range contains odd numbers or both odd and even numbers.

124 Main St. will return a 1 if "124" falls within the range and this range contains even numbers or both odd and even numbers.

Syntax

```
BooleanValue = object->IsAddressInRange2 (InputRange, LoRange,
HiRange, OddEvenFlag);
```

C

```
IntegerValue = mdStreetIsAddressInRange2 (object, InputRange,
LoRange, HiRange, OddEvenFlag);
```

COM

```
BooleanValue = object.IsAddressInRange2 (InputRange, LoRange,
HiRange, OddEvenFlag)
```

3.2.6: Auto-Complete

This function allows you to enable auto-complete for StreetData in Address Object.

GetAutoCompletion

The **GetAutoCompletion** method returns possible street address matches based off of a 5-digit US Zip code and at least one character of a street address. This method will return one possible street address with each call. The user can cycle through the entire list of possible street addresses by calling GetAutoCompletion until it returns an empty string.

Remarks

It is recommended that the user waits until at least four characters or an alphabet letter before calling GetAutoCompletion. Calling GetAutoCompletion("1", "92688", mdStreet.AutoCompletionMode, AutoCompleteSingleSuite, true) will return every street address in the zip code "92688" that starts with "1". This will be a large number of addresses and is usually not useful for the end user. It would be much more user friendly to wait until a more complete number is entered. For example, waiting until "1423" is entered and then calling **GetAutoCompletion** so the returned address list is more limited and useful.

The **GetAutoCompletion** function has the data type AutoCompletionMode that uses an enumeration to specify the handling of suites in the suggested street address.

Enumerated Value	Integer Value	Description
AutoCompleteSingleSuite	0	Show each individual suite. Ex: 123 Main Apt 1, 123 Main Apt 2
AutoCompleteRangedSuite	1	Show the suite ranges. Ex: 123 Main Apt 1-10, 123 Main Apt 20-30
AutoComplete PlaceHolderSuite	2	Use a '#' to denote where a suite should go. Ex: 123 Main #
AutoCompleteNoSuite	3	Do not show suites. Ex: 123 Main

Languages that do not use enumerations will use the equivalent integer value.

Input Parameters

The **GetAutoCompletion** function takes the following four parameters in the order given:

- *string streetAddress*: The full or a portion of a street address (123 main st)
- *string Zip*: A full 5-digit US Zip code
- *mdStreet.AutoCompletionMode enumeration*: enumeration concerning handling of suites
- *bool DPV_on_or_off*: True to return only DPV valid street addresses. False to return all street addresses

Example

```
    String SuggestAddr;
    so.ResetAutoCompletion();
    while ((SuggestAddr =
        so.GetAutoCompletion(txtAddress.Text.Trim(),
        txtZip.Text.Trim(), mode, true)) != "")
{
    //Display SuggestAddr
}
```

Syntax

```
object->GetAutoCompletion(string Address, string Zip,
mdStreet.AutoCompletionMode enumeration, bool DPV_on_or_off);
```

C

```
mdStreetGetAutoCompletion(object, string Address, string Zip,
mdStreet.AutoCompletionMode enumeration, bool DPV_on_or_off);
```

ResetAutoCompletion

The **ResetAutoCompletion** function resets the auto-completion, preparing the object for a new record.

Remarks

Call this function before calling a new record.

Calling this function ensures that the function will operate properly for the new record.

Syntax

```
object->ResetAutoCompletion();
```

C

```
mdStreetResetAutoCompletion(object);
```

CHAPTER 4
ZipData Interface

The ZipData Interface gives you access to ZIP Code related information. With this Interface, you can either validate ZIP codes or match ZIP codes to city names, latitude/longitude coordinates or county FIPS codes. You can insert the city/state information from a ZIP Code to speed up data entry.

4.1: Using the ZipData Interface

1 Create an instance of the Address Object's ZipData Interface.

```
Dim ZipPtr As New ZipCodeData
```

2 Call the **Initialize** function, passing the paths to the files `mdAddr.dat` and `mdAddr.nat`.

```
CALL Initialize WITH DataPaths RETURNING Result
IF Result <> 0 Then
   Call GetInitializeErrorString RETURNING ErrorString
   PRINT ErrorString
   End If
```

3 The ZipData Interface offers three separate operations. To retrieve a list of cities for one ZIP Code, use the **FindZip** and **FindZipNext** functions.

3a Pass a ZIP Code to the **FindZip** function. If it returns a TRUE value, then call the various functions to retrieve the return values.

```
CALL FindZip WITH Code RETURNING Result
IF Result Is TRUE THEN
   CALL GetAreaCode RETURNING AreaCode
```

```
    CALL GetAutomation RETURNING Automation
    CALL GetCity RETURNING City
    CALL GetState RETURNING State
    //
    // See below for a full list of
    // all functions returned.
    //
    CALL GetZipType RETURNING ZipType
ENDIF
```

3b Continue to call the **FindZipNext** function until it no longer returns a TRUE value.

```
WHILE FindZipNext Returns TRUE
    // Call the same functions as above.
ENDWHILE
```

4 In order to retrieve a list of ZIP Code records associated with a given city, use the **FindZipInCity** and **FindZipInCityNext** functions.

4a Pass city and state names to the **FindZipInCity** function. If it returns a TRUE value, then call the necessary functions to retrieve the values populated by this function.

```
CALL FindZipInCity WITH City, State RETURNING Result
IF Result Is TRUE
    CALL GetAutomation RETURNING Automation
    CALL GetCity RETURNING City
    //
    // See below for a full list of
    // all functions returned.
    //
    CALL GetZip RETURNING 5
    CALL GetZipType RETURNING ZipType
End If
```

4b Continue to call the **FindZipInCityNext** function until it no longer returns a TRUE value, and retrieve the return values.

```
WHILE FindZipInCityNext Returns TRUE
    // Call the same functions as above.
ENDWHILE
```

5 In order to find a list of cities in a state that match a certain pattern, such as all cities that begin with “San” in California, or simply verify that a city exists in a state, use the **FindCityInState** and **FindCityInStateNext** functions.

5a Pass a city name (or partial city name such as “San *”) and state name to the **FindCityInState** function. If it returns a TRUE value, then call the functions to retrieve the return values populated by this function.

```
CALL FindCityInState WITH City, State RETURNING Result
IF Result Is TRUE
   CALL GetCity RETURNING City
   CALL GetState RETURNING State
End If
```

5b Continue to call the **FindCityInStateNext** function until it no longer returns a TRUE value, and retrieve the returning values.

```
WHILE FindCityInStateNext Returns TRUE
   CALL GetCity RETURNING City
   CALL GetState RETURNING State
ENDWHILE
```

4.2: ZipData Functions

4.2.1: Initialize the ZipData Interface

Initializing the ZipData Interface is slightly different than the procedure for the AddressCheck Interface.

4.2.2: Retrieve ZIP Code and City Data

The three main functions of the ZipData Interface retrieve information based on ZIP Code or city names.

4.2.3: Retrieve ZIP and City Data

These functions retrieve the data returned as a result of calls to the ZipData Interface's three main functions:

4.2.4: Retrieve the Next Record

The following functions retrieve the next record, if any, after a successful call to the corresponding function. Each function returns a TRUE value until there are no more records to retrieve.

4.2.5: Computation Functions

The following functions do not require the data files to be initialized. They can be used with the GetLatitude and GetLongitude functions to calculate distances and bearing between two records.

4.2.1: Initialize the ZipData Interface

Initializing the ZipData Interface is slightly different than the procedure for the AddressCheck Interface.

SetLicenseString

This function sets the license string required to enable Address Object's complete functionality.

Remarks

Without the License String, the object will only function in DEMO mode (Nevada only).

The license string is normally set using an environment variable, either MD_LICENSE or MD_LICENSE_DEMO. Calling SetLicenseString is an alternative method for setting the license string, but applications developed for a production environment should only use the environment variable.

When using an environment variable, it is not necessary to call the **SetLicenseString** function.

For more information on setting the environment variable, see page 3 of this guide.

Input Parameters

The **SetLicenseString** function has one parameter:

LicenseString	A required string value representing the software license key.

Return Value

The **SetLicenseString** function returns a Boolean value of 0 (FALSE) or 1 (TRUE). The **SetLicenseString** function will return a FALSE Boolean value if the License String provided is incorrect.

Syntax

```
BooleanValue = object->SetLicenseString(LicenseString);
```

C

```
IntegerValue = mdZipSetLicenseString(object, LicenseString);
```

COM

```
BooleanValue = object.SetLicenseString(LicenseString)
```

GetBuildNumber

The **GetBuildNumber** function returns a string value containing the current development release build number of the Address Object.

Remarks

This is usually a three or four-character string. If you are running the Demo version of Address Object, the word DEMO will be appended to the build number.

Input Parameters

None.

Return Value

The **GetBuildNumber** function returns a string value containing the current development release build number of the Address Object.

Syntax

```
StringValue = object->GetBuildNumber();
```

C

```
StringValue = mdZipGetBuildNumber(object);
```

COM

```
StringValue = object.GetBuildNumber()
```


GetDatabaseDate

The **GetDatabaseDate** function returns the date of your address data files.

Remarks

If the **GetDatabaseDate** function is called before the **Initialize** function is called and the data files are not in the same directory as the `ADDROBJ.DLL`, this function will return a date outside the normal range of the system date, such as 1969 or 1899, depending on the system.

Input Parameters

None.

Return Value

The **GetDatabaseDate** function returns a value that represents the date of your address data files. The COM object returns a date value, while the standard object returns a string value.

Syntax

```
StringValue = object->GetDatabaseDate();
```

C

```
StringValue = mdZipGetDatabaseDate(object);
```

COM

```
DateTime = object.GetDatabaseDate()
```

Initialize

Opens the required data files and prepares the current instance for use. If the data file is in the same directory as the Component, you do not need to call the **Initialize** function. However, if there is an error, an exception will be reported instead of an error code.

Remarks

If the **Initialize** function returns a code other than zero, you can call the **GetInitializeErrorString** function to display a string describing the error.

Input Parameters

The **Initialize** function has three parameters. In VB and languages that support optional parameters, all three are considered optional. If the path is not provided, the **Initialize** function will look for the data files in the same directory as the Address Object component file.

DataPath	A string value containing the path to the location of the `mdAddr.dat` file.
NationalPath	A string value containing the path to the location of the `mdAddr.nat` file.
RegionalPath	This parameter is deprecated and no longer used. It is included for backwards compatibility. If your programming language supports optional parameters, you can ignore RegionalPath completely. If your programming language does not allow optional parameters, you must pass something to this parameter, even if it is only an empty string value.

Return Value

The **Initialize** function returns one of the following:

Code	Description
0	No error - initialization was successful.
1	Could not open the `mdAddr.dat` file.
2	Could not open the `mdAddr.nat` or `mdAddr.str` file.
4	The internal database date of the `mdAddr.dat` and `mdAddr.nat` files do not match.
5	Not all the memory buffers could be initialized.
6	Unknown error.

Syntax

```
IntegerValue = object->Initialize(DataPath, NationalPath,
RegionalPath);
```

C

```
IntegerValue = mdZipInitialize(object, DataPath, NationalPath,
RegionalPath);
```

COM

```
IntegerValue = object.Initialize(DataPath, NationalPath,
RegionalPath)
```

GetInitializeErrorString

This function returns a descriptive string to describe an error from the **Initialize** function.

Remarks

The **GetInitializeErrorString** function returns a string describing the error that occurred when the **Initialize** function failed. This is useful for putting a quick message up in a message box.

Syntax

```
StringValue = object->GetInitializeErrorString();
```

C

```
StringValue = mdZipGetInitializeErrorString(object);
```

COM

```
StringValue = object.GetInitializeErrorString()
```

GetLicenseExpirationDate

This function returns a date value corresponding to the date when the current license string expires.

Remarks

License strings issued by Melissa Data are valid a certain period of time. This function returns the date after which the current license string is no longer valid.

The COM object returns a date value, while the standard object returns a string value.

Syntax

```
StringValue = object->GetLicenseExpirationDate();
```

C

```
StringValue = mdZipGetLicenseExpirationDate(object);
```

COM

```
DateTime = object.GetLicenseExpirationDate
```

4.2.2: Retrieve ZIP Code and City Data

The three main functions of the ZipData Interface retrieve information based on ZIP Code or city names.

FindZip

This function returns the first record of the ZIP Code passed in, unless the optional GetLastLineRecord = 1 is passed as well. If this happens, it will return the last line record for the ZIP Code passed in.

Remarks

If the return value is TRUE, this function sets the following functions:

GetAreaCode	GetLastLineIndicator	GetLastLineNumber
GetPreferredLastLineNumber	GetAutomation	GetFacilityCode
GetCity	GetState	GetZip
GetCityAbbreviation	GetCountyFips	GetCountyName
GetZipType	GetLatitude	GetLongitude
GetTimeZone	GetTimeZoneCode	

A TRUE return will set all of the above-mentioned functions with the data of the record returned.

A FALSE return indicates that the **FindZip** function could not locate the ZIP Code passed in.

Input Parameters

The **FindZip** function has the following parameters:

Zip	A five-character string value that contains the ZIP Code to look up. Example: “92688”
GetLastLineRecord	Optional Boolean value. A “0” is the default value but you may also pass in a “1” to retrieve the last line record for the ZIP Code. This will return the official city name for the ZIP Code. If you set this option to “1,” you will not be able to call the **FindZipNext** function.

Return Value

The **FindZip** function returns a Boolean value of 0 (FALSE) or 1 (TRUE).

Syntax

```
BooleanValue = object->FindZip(Zip[, GetLastLineRecord]);
```

C

```
IntegerValue = mdZipFindZip (object, Zip [, GetLastLineRecord]);
```

COM

```
IntegerValue = object.FindZip (Zip [, GetLastLineRecord])
```

FindZipInCity

This function returns the first ZIP Code for the city, state combination passed to the **FindZipInCity** function.

Remarks

If the return value is TRUE, this function sets the return values for the following functions:

GetLastLineIndicator	GetLastLineNumber	GetPreferredLastLineNumber
GetAutomation	GetFacilityCode	GetZipType
GetCity	GetState	GetZip
GetCityAbbreviation	GetCountyFips	GetCountyName
GetLatitude	GetLongitude	GetTimeZone
GetTimeZoneCode		

A TRUE return will set all of the above-mentioned functions with the data of the record returned.

A FALSE return indicates that the **FindZipInCity** function could not locate the city passed in.

Input Parameters

The **FindZipInCity** function has these two parameters:

City	A 28-character (maximum) string that contains the city name to look up
State	A two-character string that contains the state the city is located in

Return Value

The **FindZipInCity** function returns a Boolean value of 0 (FALSE) or 1 (TRUE)

Syntax

```
BooleanValue = object->FindZipInCity (City, State)
```

C

```
IntegerValue = mdZipFindZipInCity (object, City, State)
```

COM

```
IntegerValue = object.FindZipInCity (City, State)
```

FindCityInState

This function returns the first city or only city (if no wildcard is used) for the city, state combination passed to this function.

Remarks

A TRUE return will set the return values of the **GetCity** and **GetState** functions with the data of the record returned.

A FALSE return indicates that the **FindCityInState** function could not locate the city passed to the function.

Input Parameters

The **FindCityInState** function has these two parameters:

City	A 28-character maximum string that contains the city name to look up. For example: “Los Angeles” or a WildCard (*) such as “Los A*”
State	A two-character string that contains the state abbreviation where the city is located in (Wildcard not allowed in this function).

Return Value

The **FindCityInState** function returns a Boolean value of 0 (FALSE) or 1 (TRUE).

Syntax

```
BooleanValue = object->FindCityInState(City, State);
```

C

```
IntegerValue = mdZipFindCityInState(object, City, State);
```

COM

```
IntegerValue = object.FindCityInState(City, State)
```

4.2.3: Retrieve ZIP and City Data

These functions retrieve the data returned as a result of calls to the ZipData Interface's three main functions:

- FindZip
- FindZipInCity
- FindCityInState

GetAreaCode

This function returns the area code associated with the ZIP Code after a successful call to the **FindZip** and **FindZipNext** functions.

Remarks

The **GetAreaCode** function returns a three-character string value after a call to the **FindZip** and **FindZipNext** functions.

> *If a ZIP Code has more than one area code assigned to it, the dominant area code will be returned.*

Syntax

```
StringValue = object->GetAreaCode();
```

C

```
StringValue = mdZipGetAreaCode(object);
```

COM

```
StringValue = object.AreaCode
```

GetAutomation

This function returns the carrier route rate mail indicator after a successful call to the **FindZip**, **FindZipNext** , **FindZipInCity**, or **FindZipInCityNext** function.

Remarks

The return value is an indicator that specifies the following conditions for letter-size carrier route sorted mail:

Code	Explanation
A	Carrier route sortation rates apply for this ZIP Code and merging is permitted.

Code	Explanation
B	Carrier route sortation rates apply for this ZIP Code and merging is not permitted.
C	Carrier route sortation rates do not apply for this ZIP Code and merging is permitted.
D	Carrier route sortation rates do not apply for this ZIP Code and merging is not permitted.

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetAutomation();
```

C

```
StringValue = mdZipGetAutomation(object);
```

COM

```
StringValue = object.Automation
```


GetCity

This function returns the full city name associated with the input information.

Remarks

The **GetCity** function returns a 28-character (maximum) string value after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, **FindCityInState**, **FindCityInStateNext**, or **FindZipInCityNext** function.

The return value of the **GetCity** function will contain the city name for the input ZIP Code. If the city name is longer than 13 letters, the **GetCityAbbreviation** function will also return a value.

Syntax

```
StringValue = object->GetCity();
```

C

```
StringValue = mdZipGetCity(object);
```

COM

```
StringValue = object.City
```

GetState

This function returns the state name abbreviation from each ZIP Code record.

Remarks

The **GetState** function returns a two-character string value containing the two-letter abbreviation for the state name, after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, **FindCityInState**, **FindCityInStateNext**, or **FindZipInCityNext** function.

Syntax

```
StringValue = object->GetState();
```

C

```
StringValue = mdZipGetState(object);
```

COM

```
StringValue = object.State
```

GetZip

This function returns the five-digit ZIP Code for the current record.

Remarks

The **GetZip** function returns a five-character string containing the ZIP Code for the current record, after a successful call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

Syntax

```
StringValue = object->GetZip();
```

C

```
StringValue = mdZipGetZip(object);
```

COM

```
StringValue = object.
```

GetCityAbbreviation

This function returns the official 13-letter city name abbreviation after a successful call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

Remarks

If the return value of the **GetCity** function is longer than 13 letters, the **GetCityAbbreviation** function will return the official abbreviation the Post Office has associated with that city or municipality name. For example, for “Rancho Santa Margarita,” the **GetCityAbbreviation** function will return the abbreviation “Rcho Sta Marg.”

If the return value of the **GetCity** function is 13 letters long or shorter, the **GetCityAbbreviation** function will return the full city or municipality name.

Syntax

```
StringValue = object->GetCityAbbreviation();
```

C

```
StringValue = mdZipGetCityAbbreviation(object);
```

COM

```
StringValue = object.CityAbbreviation
```


GetCountyFips

This function returns the county FIPS code for each ZIP Code record.

Remarks

The **GetCountyFips** function is a five-character string value set by a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

The Federal Information Processing Standard (FIPS) is a five-digit code defined by the U.S. Bureau of Census. The first two digits are the ZIP Code and the last three indicate the county within the state.

For Example: “06037” is the County FIPS for Los Angeles, CA (“06” is the ZIP Code for California and “037” is the county code for Los Angeles).

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetCountyFips();
```

C

```
StringValue = mdZipGetCountyFips(object);
```

COM

```
StringValue = object.CountyFips
```

GetCountyName

This function returns the name of the county associated with the input ZIP Code.

Remarks

The **GetCountyName** function returns a 25-character string after a successful call to the **FindZip** or **FindZipNext** function.

Syntax

```
StringValue = object->GetCountyName();
```

C

```
StringValue = mdZipGetCountyName(object);
```

COM

```
StringValue = object.CountyName
```

GetFacilityCode

This function returns the facility code for each ZIP Code record returned.

Remarks

After a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function, the **GetFacilityCode** function returns a one-character string value that indicates the type of postal facility:

Code	Type
A	Airport Mail Facility
B	Branch
C	Community Post Office
D	Area Distribution Center
E	Sectional Center Facility
F	Delivery Distribution Center
G	General Mail Facility
K	Bulk Mail Facility
M	Money Order Unit
N	Community or Place name
P	Post Office
S	Station
U	Urbanization (Used in Puerto Rico)
X	Vanity name (Should not be used)

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

C++

```
StringValue = object->GetFacilityCode();
```

C

```
StringValue = mdZipGetFacilityCode(object);
```

COM

```
StringValue = object.FacilityCode
```

GetLastLineIndicator

This function returns the last line indicator for the returned ZIP Code data.

Remarks

The **GetLastLineIndicator** function returns a one-character string value after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function. An "L" in this field indicates that the city name is the official U. S. Postal Service name for the ZIP Code (Only one record per ZIP Code is coded with an "L").

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetLastLineIndicator();
```

C

```
StringValue = mdZipGetLastLineIndicator(object);
```

COM

```
StringValue = object.LastLineIndicator
```

GetLastLineNumber

This function returns the last line number of each ZIP record returned.

Remarks

The **GetLastLineNumber** function returns a six-character string value after a call to **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function. This number is used with address matching to break ties on certain records using the city name.

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetLastLineNumber();
```

C

```
StringValue = mdZipGetLastLineNumber(object);
```

COM

```
StringValue = object.LastLineNumber
```

GetLatitude

This function returns the latitude as a string for the geographic center of a ZIP Code.

Remarks

Latitude is the geographic coordinate of a point measured in degrees north or south of the equator.

The **GetLatitude** function returns a seven-character string value after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function. The latitude returned will be accurate to 4 decimal places and will always be positive to indicate a latitude in the United States, which is north of the equator.

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetLatitude();
```

C

```
StringValue = mdZipGetLatitude(object);
```

COM

```
StringValue = object.Latitude
```


GetLongitude

This function returns the longitude as a string for the geographic center of a ZIP Code.

Remarks

Longitude is the geographic coordinate of a point measured in degrees east or west of the Greenwich Meridian.

The **GetLongitude** function returns the longitude of the record after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function. It is accurate to 6 decimal places and the negative sign is used to indicate a longitude in the United States.

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetLongitude();
```

C

```
StringValue = mdZipGetLongitude(object);
```

COM

```
StringValue = object.Longitude
```

GetMsa (Deprecated)

This function returns the MSA for each ZIP Code record returned.

Because PMSA and MSA information is no longer updated, **this function has been deprecated.**

Remarks

The **GetMsa** function returns a four-digit value after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

The Office of Management and Budget defines the Metropolitan Statistical Area (MSA) as one or more counties forming a large population with adjacent communities and having a high degree of social and economic integration.

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetMsa();
```

C

```
StringValue = mdZipGetMsa(object);
```

COM

```
StringValue = object.Msa
```

GetPmsa (Deprecated)

This function returns the PMSA associated with each ZIP Code record.

Because PMSA and MSA information is no longer updated, **this function has been deprecated.**

Remarks

The **GetPmsa** function returns a four-digit value after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

The Office of Management and Budget defines the Primary Metropolitan Statistical Area (PMSA) for regions that contain a population of more than one million.

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetPmsa();
```

C

```
StringValue = mdZipGetPmsa(object);
```

COM

```
StringValue = object.Pmsa
```

GetPreferredLastLineNumber

This function returns the preferred last line number for each ZIP Code record.

Remarks

The **GetPreferredLastLineNumber** function returns a six-character string value containing the preferred last line number for the city name, after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

This is similar to the **GetLastLineNumber** function mentioned earlier. If the preferred last line number is the same as the last line number, this city name is the one the Post Office will recognize as the official name for that ZIP Code.

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetPreferredLastLineNumber();
```

C

```
StringValue = mdZipGetPreferredLastLineNumber(object);
```

COM

```
StringValue = object.PreferredLastLineNumber
```

GetTimeZone

This function returns the time zone for each ZIP Code record.

Remarks

All Melissa Data products express time zones in UTC (Coordinated Universal Time).

The **GetTimeZone** function returns a 20-character (maximum) string value after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

These are the strings that can be returned:

“Military”	“Mountain Time”	“Samoa Time”
“Atlantic Time”	“Pacific Time”	“Marshall Islands Time”
“Eastern Time”	“Alaska Time”	“Guam Time”
“Central Time”	“Hawaii Time”	“Palau Time”

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetTimeZone();
```

C

```
StringValue = mdZipGetTimeZone(object);
```

COM

```
StringValue = object.TimeZone
```

GetTimeZoneCode

This function returns a one or two-digit code for the time zone that contains the specified ZIP Code.

Remarks

The **GetTimeZoneCode** function returns a one or two-digit number representing the time zone for the record returned after a call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

These are the possible values for the **GetTimeZone** function:

Code	Zone	Code	Zone
0	Military (APO or FPO)	9	Alaska Time
4	Atlantic Time	10	Hawaii Time
5	Eastern Time	11	Samoa Time
6	Central Time	12	Marshall Island Time
7	Mountain Time	14	Guam Time
8	Pacific Time	15	Palau Time

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetTimeZoneCode();
```

C

```
StringValue = mdZipGetTimeZoneCode(object);
```

COM

```
StringValue = object.TimeZoneCode
```

GetZipType

This function returns the ZIP Code type for each record returned.

Remarks

The **GetZipType** function returns a one-character string after a successful call to the **FindZip**, **FindZipNext**, **FindZipInCity**, or **FindZipInCityNext** function.

The return value of the **GetZipType** function defines the class of ZIP Code for delivery purposes.

Code	Explanation
P	A ZIP Code used only for PO Boxes.
U	Unique: A ZIP Code assigned to an organization or government institution such as the IRS.
M	Military: A ZIP Code assigned to an APO/FPO.
Empty	A standard ZIP Code.

This function does not return a value when the ZipData interface is used with a Canadian postal code.

Syntax

```
StringValue = object->GetZipType();
```

C

```
StringValue = mdZipGetZipType(object);
```

COM

```
StringValue = object.ZipType
```

4.2.4: Retrieve the Next Record

The following functions retrieve the next record, if any, after a successful call to the corresponding function. Each function returns a TRUE value until there are no more records to retrieve.

FindZipNext

This function retrieves the next record of the ZIP Code passed into the **FindZip** function.

Remarks

If the return value is TRUE, this function sets the return values for the same functions as the **FindZip** function. See page 175 for a list of these functions.

A TRUE return will set all of the above-mentioned functions with the data of the record returned.

A FALSE return indicates that the **FindZipNext** function could not locate another record for the ZIP Code passed in.

Input Parameters

The **FindZipNext** function operates on the ZIP passed to the **FindZip** function. You MUST call the **FindZip** function before you call the **FindZipNext** function.

> *If you passed a "1" for the optional GetLastLineRecord parameter in the **FindZip** function, you cannot call the **FindZipNext** function.*

Return Value

The **FindZipNext** function returns a Boolean value of 0 (FALSE) or 1 (TRUE).

Syntax

```
BooleanValue = object->FindZipNext();
```

C

```
IntegerValue = mdZipFindZipNext(object);
```

COM

```
IntegerValue = object.FindZipNext()
```

FindZipInCityNext

This function retrieves the next ZIP Code for the city, state combination passed to the **FindZipInCity** function.

Remarks

If the return value is TRUE, this function sets the same functions as the **FindZipInCity** function. See page 176 for a list of these functions.

A TRUE return will set all of the above-mentioned functions with the data of the record returned.

A FALSE return indicates that the **FindZipInCityNext** function could not locate any more ZIP Codes for the city passed in.

Input Parameters

The **FindZipInCityNext** function operates on the ZIP passed to the **FindZipInCity** function. You MUST call the **FindZipInCity** function before you call the **FindZipInCityNext** function.

Return Value

The **FindZipInCityNext** function returns a Boolean value of 0 (FALSE) or 1 (TRUE).

Syntax

```
BooleanValue = object->FindZipInCityNext();
```

C

```
IntegerValue = mdZipFindZipInCityNext(object);
```

COM

```
IntegerValue = object.FindZipInCityNext()
```

FindCityInStateNext

This function retrieves the next city for the city, state combination passed to the **FindCityInState** function.

Remarks

A TRUE return indicates that the function has set the return values of the **GetCity** and **GetState** functions with the values from the next record returned.

A FALSE return indicates that the **FindCityInStateNext** function could not locate any more cities for the city passed to **FindCityInState**

Input Parameters

The **FindCityInStateNext** function operates on the city passed to the **FindCityInState** function. You MUST call the **FindCityInState** function before you call the **FindCityInStateNext** function.

Return Value

The **FindCityInStateNext** function returns a Boolean value of 0 (FALSE) or 1 (TRUE).

Syntax

```
BooleanValue = object->FindCityInStateNext();
```

C

```
IntegerValue = mdZipFindCityInStateNext(object);
```

COM

```
IntegerValue = object.FindCityInStateNext()
```

4.2.5: Computation Functions

The following functions do not require the data files to be initialized. They can be used with the **GetLatitude** and **GetLongitude** functions to calculate distances and bearing between two records.

ComputeBearing

The **ComputeBearing** function returns a bearing in degrees (0 to 360) representing the compass direction from point 1 to point 2.

North is 0 degrees, East is 90, South is 180, and West is 270.

Remarks

You do not have to call the **Initialize** function before the **ComputeBearing** function is called.

Input Parameters

This function accepts four double-precision floating point numbers.

- **lat1** — Latitude for point 1 [In Degrees (90 to –90)]
- **long1** — Longitude for point 1 [In Degrees (180 to –180)]
- **lat2** — Latitude for point 2 [In Degrees (90 to –90)]
- **long2** — Longitude for point 2 [In Degrees (180 to –180)]

Negative longitudes lie west of the Greenwich Meridian, positive longitudes to the east. Longitudes for the continental U.S. are in the range from -65 to -125 degrees.

Return Value

The **ComputeBearing** function returns the bearing based on the input latitudes and longitudes. If an invalid latitude or longitude is entered, a value of 999 will be returned.

Syntax

```
DoubleFloat = object->ComputeBearing(lat1, long1, lat2, long2);
```

C

```
DoubleFloat = mdZipComputeBearing(object, lat1, long1, lat2,
long2);
```

COM

```
DoubleFloat = object.ComputeBearing(lat1, long1, lat2, long2)
```

ComputeDistance

The **ComputeDistance** function returns a distance in miles between point 1 and point 2.

Remarks

You do not have to call the **Initialize** function before the **ComputeDistance** function is called.

Input Parameters

This function accepts four double-precision floating point numbers.

- **lat1** — Latitude for point 1 [In Degrees (90 to –90)]
- **long1** — Longitude for point 1 [In Degrees (180 to –180)]
- **lat2** — Latitude for point 2 [In Degrees (90 to –90)]
- **long2** — Longitude for point 2 [In Degrees (180 to –180)]

Negative longitudes lie west of the Greenwich Meridian, positive longitudes to the east. Longitudes for the continental U.S. are in the range from -65 to -125 degrees.

Return Value

The **ComputeDistance** function returns the distance between two points based on the input latitudes and longitudes. If an invalid latitude or longitude is entered, a value of 9999 will be returned.

Syntax

```
DoubleFloat = object->ComputeDistance (lat1, long1, lat2,
long2);
```

C

```
DoubleFloat = mdZipComputeDistance (object lat1, long1, lat2,
long2);
```

COM

```
DoubleFloat = object.ComputeDistance (lat1, long1, lat2, long2)
```

CHAPTER 5
Parse Interface

The Parse Interface allows a user to separate a street address into its individual components. There are nine possible components that make up an address. They are Range, PreDirection, StreetName, StreetSuffix, PostDirection, SuiteName, SuiteNumber, PrivateMailboxName, and PrivateMailboxNumber.

There are several reasons to use address parsing. You may want to search our StreetData database to manually correct records our AddressCheck Interface was unable to code, or you may need to sort addresses in your list by street name and need a fast and easy function to remove the street name from an embedded address.

Addresses can usually be parsed correctly on the first try. For example, the address “100 Main St” can easily be separated so the “100” is the range, “Main” is the street name, and “St” is the street suffix. However, address parsing is not an exact science and some addresses are more ambiguous and require more attention.

The address parser employs a technique known as multiple pass parsing to handle any ambiguous addresses. If the address is not parsed correctly the first time, the parse can be called a second or third time. Each time it is called it will rearrange the tokens into a new order to try to get a match.

Let's look at an example. Consider the address "100 N Ave C." This address can be parsed correctly in any of the following ways:

Range	Pre	Street Name	Street Suffix	Suite Name
100	N	Ave C		
100	N	Avenue C		
100	N	Ave		C
100	N	Avenue		C
100		N	Ave	C
100		North	Ave	C

Any of the above combinations can be correct, and all probably exist in one form somewhere in the country.

The **Parse** function will return the first parse shown here and it will be correct 90 to 95% of the time. If you need a second look at the address, repeated calls to the **ParseNext** function will produce the additional combinations that are listed above.

5.1: Using the Parse Interface

1 Create an instance of the Address Object's Parse Interface.

```
Set parsPtr As New Instance of Parse
```

2 Call the **Parse** function, passing the string value containing the address to be parsed. Check the return value to verify that Address Object was able to parse the address string. If it was, the return values of the interface's functions have been populated with the parsed address data.

```
Call Parse WITH StreetAddress RETURNING ParseResult
IF ParseResult = 0 Then
    PRINT "Unable To Parse Address"
ELSE
    CALL GetRange RETURNING Range
    CALL GetPreDirection RETURNING PreDirection
    CALL GetStreetName RETURNING StreetName
    CALL GetSuffix RETURNING Suffix
    CALL GetPostDirection RETURNING PostDirection
    CALL GetSuiteName RETURNING SuiteName
    CALL GetSuiteNumber RETURNING SuiteNum
    CALL GetPrivateMailboxName RETURNING PrivateMailboxName
    CALL GetPrivateMailboxNumber RETURNING PrivateMailboxNum
    CALL GetGarbage Return GarbageText
ENDIF
```

3 The parsed strings should be placed in the matching function (Street number in Range, the name of the street in StreetName, and so on). If the parsed data does not correctly match the return function names, which can happen if the submitted address does not follow a standard format, you can call the **ParseNext** function to re-parse the submitted string, returning the parsed data in a different order.

If the **ParseNext** function returns "0," there were no other possible parses for the submitted data.

```
CALL ParseNext RETURNING Result
While Result Is 0
    CALL GetRange RETURNING Range
    CALL GetPreDirection RETURNING PreDirection
    CALL GetStreetName RETURNING StreetName
    CALL GetSuffix RETURNING Suffix
    CALL GetPostDirection RETURNING PostDirection
    CALL GetSuiteName RETURNING SuiteName
    CALL GetSuiteNumber RETURNING SuiteNum
    CALL GetPrivateMailboxName RETURNING PrivateMailboxName
    CALL GetPrivateMailboxNumber RETURNING PrivateMailboxNum
    CALL GetGarbage Return GarbageText
    CALL ParseNext RETURNING Result
ENDWHILE
```

4 If you have a string value containing "City, State ZIP" data, pass this to the **LastLineParse** function.

```
CALL LastLineParse WITH "Rancho Santa Margarita, CA 92688"

CALL GetCity RETURNING City
CALL GetState RETURNING State
CALL GetZip RETURNING Zip5
CALL GetPlus4 RETURNING Plus4
```

5.2: Parse Interface Functions

5.2.1: Initialize the Parse Interface

The Parse Interface does not need to be initialized in the same manner as the other interfaces, nor does it require a license string to use.

5.2.2: Parse an Address

These two functions parse a submitted street address and then return all possible variations on how the address can be parsed.

5.2.3: Retrieve the Parsed Address Components

These functions return the parsed components of a street address after a call to the Parse, ParseCanadian, or ParseNext function.

5.2.4: Parse Last Lines

Use these functions to break a single string value into city, state, and ZIP Code.

5.2.1: Initialize the Parse Interface

The Parse Interface does not need to be initialized in the same manner as the other interfaces, nor does it require a license string to use.

GetBuildNumber

The **GetBuildNumber** function returns a string value containing the current development release build number of the Address Object.

Remarks

This is usually a three or four-character string. If you are running the Demo version of Address Object, the word DEMO will be appended to the build number.

Input Parameters

None.

Return Value

The **GetBuildNumber** function returns a string value containing the current development release build number of the Address Object.

Syntax

```
StringValue = object->GetBuildNumber();
```

C

```
StringValue = mdParseGetBuildNumber(object);
```

COM

```
StringValue = object.GetBuildNumber()
```

5.2.2: Parse an Address

These two functions parse a submitted street address and then return all possible variations on how the address can be parsed.

Parse

The **Parse** function will separate an input address string into its individual components and populate the return values of the appropriate functions with those string values.

Input Parameters

The **Parse** function has one parameter:

AddressToParse	A string that contains the full street address to be parsed, such as "123 N Main St. E Apt 3."

Return Value

The **Parse** function always returns a Boolean value of 1 to indicate TRUE.

Use the following functions to retrieve the parsed address:

GetRange	GetPreDirection
GetStreetName	GetSuffix
GetPostDirection	GetSuiteName
GetSuiteNumber	GetPrivateMailboxName
GetPrivateMailboxNumber	GetGarbage

Syntax

```
BooleanValue = object->Parse(AddressToParse);
```

C

```
IntegerValue = mdParseParse(object, AddressToParse);
```

COM

```
BooleanValue = object.Parse(AddressToParse)
```

ParseCanadian

Canada Only — This function is identical to the **Parse** function, except that it must be used when parsing Canadian street address.

Remarks

The usage for this function is the same as it is for the **Parse** function, except that it also returns values to the following functions:

GetRouteService	GetLockBox
GetDeliveryInstallation	

Syntax

```
BooleanValue = object->ParseCanadian(AddressToParse);
```

C

```
IntegerValue = mdParseParseCanadian(object, AddressToParse);
```

COM

```
BooleanValue = object.ParseCanadian(AddressToParse)
```

ParseNext

The **ParseNext** function will juggle the address components separated in a first call to either the **Parse** or **ParseCanadian** function. This is sometimes useful when trying to match addresses and the first parse failed to find a match.

Input Parameters

The **ParseNext** function has no parameters, it can only be called after a call to the **Parse** or **ParseCanadian** function.

ParseNext acts on the input AddressToParse string passed into the **Parse** or **ParseCanadian** function.

Return Value

The **ParseNext** function returns a Boolean value of 0 (FALSE) to indicate there are no more parses, or a Boolean value of 1(TRUE) to indicate another possible parse was returned.

Use the following functions to retrieve the parsed address:

GetRange	GetPreDirection
GetStreetName	GetSuffix

GetPostDirection	GetSuiteName
GetSuiteNumber	GetPrivateMailboxName
GetPrivateMailboxNumber	GetGarbage

For Canadian addresses, you will also need the following functions:

GetRouteService	GetLockBox
GetDeliveryInstallation	

Syntax

```
BooleanValue = object->ParseNext();
```

C

```
IntegerValue = mdParseParseNext(object);
```

COM

```
BooleanValue = object.ParseNext()
```

5.2.3: Retrieve the Parsed Address Components

These functions return the parsed components of a street address after a call to the **Parse**, **ParseCanadian**, or **ParseNext** function.

GetRange

This function returns the delivery number of a parsed address.

Remarks

The **GetRange** function returns a 10-character maximum string value that contains the range portion of the address string passed to the **Parse** function.

Syntax

```
StringValue = object->GetRange();
```

C

```
StringValue = mdParseGetRange(object);
```

COM

```
StringValue = object.Range
```

GetPreDirection

This function returns the geographic directional that precedes the street name.

Remarks

The **GetPreDirection** function is a two-character string value set by a call to the **Parse** function.

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

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

Syntax

```
StringValue = object->GetPreDirection();
```

C

```
StringValue = mdParseGetPreDirection(object);
```

COM

```
StringValue = object.PreDirection
```

GetStreetName

This function returns the name of the street from a parsed address.

Remarks

The **GetStreetName** function returns a maximum 28-character string set by a call to the **Parse** function.

Syntax

```
StringValue = object->GetStreetName();
```

C

```
StringValue = mdParseGetStreetName(object);
```

COM

```
StringValue = object.StreetName
```

GetSuffix

This function returns the suffix portion of a parsed address.

Remarks

The **GetSuffix** function returns a four-character maximum string value containing the abbreviated suffix of the full address string passed into the **Parse** function.

When street names are used more than once within a city or area, street suffixes are often used to distinguish them. For example, if there is a Main Street but also a Main

Avenue, this field will hold either the “St” or “Ave” suffix, depending on which one is being referred to.

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

Syntax

```
StringValue = object->GetSuffix();
```

C

```
StringValue = mdParseGetSuffix(object);
```

COM

```
StringValue = object.Suffix
```

GetPostDirection

This function returns any geographic directional that follows the street name.

Remarks

The **GetPostDirection** function returns a two-character maximum string value containing the abbreviated post-direction of the full address string passed into the **Parse** function. Directions such as “North” will be changed to “N” before they are returned by this function.

Syntax

```
StringValue = object->GetPostDirection();
```

C

```
StringValue = mdParseGetPostDirection(object);
```

COM

```
StringValue = object.PostDirection
```

GetSuiteName

This function returns the name of the secondary unit of a parsed address.

Remarks

The **GetSuiteName** function returns a four-character maximum string value containing the suite name portion from the full address string passed into the **Parse** function.

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

Syntax

```
StringValue = object->GetSuiteName();
```

C

```
StringValue = mdParseGetSuiteName(object);
```

COM

```
StringValue = object.SuiteName
```

GetSuiteNumber

This function returns the range of any secondary unit of a parsed address.

Remarks

The **GetSuiteNumber** function returns a six-character maximum string value containing the suite number of the full address string passed into the **Parse** function.

Syntax

```
StringValue = object->GetSuiteNumber();
```

C

```
StringValue = mdParseGetSuiteNumber(object);
```

COM

```
StringValue = object.SuiteNumber
```

GetPrivateMailboxName

This function returns the private mailbox name portion of a parsed address.

Remarks

The Parse Interface will not identify an address as a PMB unless the string passed to the **Parse** function contains the “PMB” abbreviation.

Syntax

```
StringValue = object->GetPrivateMailBoxName();
```

C

```
StringValue = mdParseGetPrivateMailBoxName(object);
```

COM

```
StringValue = object.PrivateMailBoxName
```

GetPrivateMailboxNumber

This function returns the private mailbox number portion of a parsed address.

Remarks

The Parse Interface will not identify an address as a PMB unless the string passed to the **Parse** function contains the “PMB” abbreviation.

Syntax

```
StringValue = object->GetPrivateMailBoxNumber();
```

C

```
StringValue = mdParseGetPrivateMailBoxNumber(object);
```

COM

```
StringValue = object.PrivateMailBoxNumber
```

GetRouteService

Canada Only — This function returns the route service information from a parsed Canadian street address.

Remarks

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

Syntax

```
StringValue = object->GetRouteService();
```

C

```
StringValue = mdParseGetRouteService(object);
```

COM

```
StringValue = object.RouteService
```

GetLockBox

Canada Only — This function returns the Lock Box information from a parsed Canadian street address.

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

```
StringValue = object->GetLockBox();
```

C

```
StringValue = mdParseGetLockBox(object);
```

COM

```
StringValue = object.LockBox
```

GetDeliveryInstallation

Canada Only — This function returns the Delivery Installation information from a parsed Canadian street address.

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

```
StringValue = object->GetDeliveryInstallation();
```

C

```
StringValue = mdParseGetDeliveryInstallation(object);
```

COM

```
StringValue = object.DeliveryInstallation
```

GetGarbage

This function returns any garbage characters left over after a street address has been parsed.

Remarks

The **GetGarbage** function is a 50-character maximum string value set by a call to the **Parse** function containing any characters or tokens that do not belong to the address string passed into the **Parse** function. Common items found in this field are community center names, names of individuals, additional delivery instructions, and so on.

Example

“West Wing” or “Beaumont Main Office”

Syntax

```
StringValue = object->GetGarbage();
```

C

```
StringValue = mdParseGetGarbage(object);
```

COM

```
StringValue = object.Garbage
```

5.2.4: Parse Last Lines

Use these functions to break a single string value into city, state, and ZIP Code.

LastLineParse

The **LastLineParse** function parses a single string value containing a city, state, and ZIP Code string into separate string values.

Remarks

After calling this function, call the **GetCity**, **GetState**, **GetZip**, and **GetPlus4** functions to retrieve the parsed values.

Input Parameters

This function has a single input parameter:

LastLine	A single city, state, and ZIP Code string, such as "Rancho Santa Margarita, CA 92688-2112."

This function can be used for both U.S. and Canadian cities.

Syntax

```
void = object->LastLineParse(LastLine);
```

C

```
void = mdParseLastLineParse(object, LastLine);
```

COM

```
object.LastLineParse(LastLine)
```

GetCity

This function returns the city name as parsed from the **LastLineParse** function.

Remarks

The **GetCity** function returns the city portion of a parsed Last Line string after a call to the **LastLineParse** function.

Syntax

```
StringValue = object->GetCity();
```

C

```
StringValue = mdParseGetCity(object);
```

COM

```
StringValue = object.City
```

GetState

This function returns the state name abbreviation as parsed from the **LastLineParse** function.

Remarks

The **GetState** function returns the state portion of a parsed Last Line string after a call to the **LastLineParse** function.

Syntax

C++

```
StringValue = object->GetState();
```

C

```
StringValue = mdParseGet(object);
```

COM

```
StringValue = object.
```

GetZip

This function returns the ZIP Code as parsed from the **LastLineParse** function.

Syntax

```
StringValue = object->GetZip();
```

C

```
StringValue = mdParseGet(object);
```

COM

```
StringValue = object.
```

GetPlus4

This function returns the Plus4 extension of the ZIP Code from a successful call to the **LastLineParse** function.

Remarks

The **LastLineParse** function does not do any address checking, therefore if the string passed to the function does not contain a recognizable four-digit ZIP add-on, this function will not return a value.

Also, if the **LastLineParse** function is used with a Canadian municipality, province and postal code, this function will not return a value.

Syntax

```
StringValue = object->GetPlus4();
```

C

```
StringValue = mdParseGetPlus4(object);
```

COM

```
StringValue = object.Plus4
```

License Agreement

1. NOTICE. MELISSA DATA CORPORATION IS WILLING TO LICENSE THE ENCLOSED SOFTWARE TO YOU, ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS LICENSE AGREEMENT. PLEASE READ THIS LICENSE AGREEMENT CAREFULLY BEFORE OPENING THE SEALED DISK PACKAGE. BY OPENING THIS PACKAGE (OR IN THE CASE OF DOWNLOADED SOFTWARE, YOU REQUEST UNLOCKING CODE FROM THE PUBLISHER) YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THESE TERMS WE ARE UNWILLING TO LICENSE THE SOFTWARE TO YOU, AND YOU SHOULD NOT OPEN THE DISK PACKAGE. IN SUCH CASE, PROMPTLY RETURN THE UNOPENED DISK PACKAGE AND ALL OTHER MATERIAL IN THIS PACKAGE ALONG WITH PROOF OF PAYMENT, TO THE AUTHORIZED DEALER FROM WHOM YOU OBTAINED IT FOR A FULL REFUND OF THE PRICE YOU PAID.
2. **Ownership and License.** This is a license agreement and NOT an agreement for sale. We continue to own the copy of the Software (including, but not limited to, object code, dynamic link libraries, and sample programs, together with the accompanying documentation contained in this package and all other copies that you are authorized by this Agreement to make collectively known as “Software”). Your rights to use the Software are specified in this Agreement, and we retain all rights not expressly granted to you in this Agreement. This Software is protected by U.S. copyright laws and international treaties. Nothing in this Agreement constitutes a waiver of our rights under U.S. Copyright law or any other federal or law or international treaty.

3. **Permitted Uses.** You are granted the following rights to the Software:

   (a) **Right to Install and Use.**

   (1) **Standalone Computer** - Single Installation You may install and use the Software on the hard disk drive of any single compatible computer that you own. However, you may not under any circumstances have the Software installed onto the hard drives of two or more computers at the same time, (nor may you install the Software onto the hard disk drive of one computer and then use the original CD-ROM on another computer). If you wish to use the Software on more than one computer, you must either erase the Software from the first hard drive before you install it onto a second hard drive, or else license an additional copy of the Software for each additional computer on which you want to use it.

   (2) **Network Use** - If the single computer on which you install the Software is a network or Internet server, you may use the Software on any computer attached to the network, provided that it is only installed on the server. You may install and use this Software on a single file server regardless of the number of workstations attached to the network.

   (b) **Right to Copy.** You may copy the Software for backup and archival purposes, provided that the original and each copy is kept in your possession, and that your installation and use of the Software does not exceed that allowed in part (a) above.

   (1) Solely with the respect to the manual and Help files, you may make an unlimited number of copies (either in hard-copy or electronic form), provided that such copies shall be used only for internal purposes and are not republished or distributed beyond the licensee's premises.

   (2) Copy, bundle, or redistribute the DEMO software with any commercial product (including books, DVD-ROM, computer hardware, or software products). Your promotional and/or packaging materials must clearly disclose that the Software is copyrighted software of Melissa Data, that no charge is made by Melissa Data or you for it, and that it is not a fully supported commercial version.

   (c) **Right to Modify.** You may modify the Software and/or merge it into another computer program to the extent necessary for your own use on (a single computer or server as specified above); however, any portion of the Software merged into another computer program will continue to be subject to the terms of this Agreement. You may use and modify the source code version of those Software portions that the documentation identifies as sample code ("SAMPLE CODE"), provided you do not distribute the SAMPLE CODE or any modified version of the SAMPLE CODE, in source form.

   (d) **Right to Transfer.** You may not rent, lend, or lease this Software. However, you may transfer this license to use the Software to another party on a permanent basis by transferring this copy of the License Agreement, at least one unaltered copy of the Software, and all documentation. You must, at the

same time, either transfer to the other party or destroy all your other copies of the Software or destroy all of your copies. Such transfer of possession terminates your license from us. Such other party shall be licensed under the terms of this Agreement upon its acceptance of this Agreement by its initial use of the Software. If you transfer the Software, you must remove the Software from your hard disk and you may not retain any copies of the Software for your own use.

4. **Prohibited Uses.** You may not, without written permission from us:

   (a) Use, copy, modify, merge, or transfer copies of the Software or documentation except as provided in this Agreement;

   (b) Use any backup or archival copies of the Software (or allow someone else to use such copies) for any purpose other than to replace the original copy in the event it is destroyed or becomes defective;

   (c) Disassemble, decompile or reverse engineer, or in any manner decode the Software for any reason;

   (d) Distribute, Sublicense, lease, or rent the Software, Data files and/or Dynamic Link Libraries of the Software.

   (e) Expose the interfaces of the Software through your application (e.g. an OCX, DLL, class library, etc.).

5. **Limited Warranty.** We make the following limited warranties, for a period of 180 days from the date you acquired the Software from us.

   (a) **Media.** The disks and documentation in this package will be free from defects in materials and workmanship under normal use. If the disks or documentation fail to conform to this warranty, you may, as your sole and exclusive remedy, obtain a replacement free of charge if you return the defective disk or documentation to us with a dated proof of purchase.

   (b) **Software.** The Software in this package will materially conform to the documentation that accompanies it. If the Software fails to operate in accordance with this warranty, you may, as your sole and exclusive remedy, return all of the Software and the documentation to the authorized dealer from whom you acquired it, along with a dated proof of purchase, specifying the problem, and we will provide you with a new version of the Software or a full refund at our election.

   (c) WARRANTY DISCLAIMER. WE DO NOT WARRANT THAT THIS SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT ITS OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE. WE EXCLUDE AND EXPRESSLY DISCLAIM ALL EXPRESS AND IMPLIED WARRANTIES NOT D HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

6. **Termination.** This license and your right to use this Software automatically terminate if you fail to comply with any provisions of this Agreement, destroy the copies of the Software in your possession, or voluntarily return the Software to us. Upon termination you will destroy all copies of the Software and documentation. Otherwise, the restrictions on your rights to use the Software will expire upon expiration of the copyright to the Software.
7. **Miscellaneous Provisions.** This Agreement will be governed by and construed in accordance with the substantive laws of California. This is the entire agreement between us relating to the contents of this package, and supersedes any prior purchase order, communications, advertising or representations concerning the contents of this package. No change or modification of this Agreement will be valid unless it is in writing, and is signed by us.