Data Quality Suite

AddressObject EmailObject NameObject PhoneObject

Multiplatform

MELISSA DATA®
Your Partner in Data Quality

Data Quality Suite

Quick Start Guide for .NET Developers

COPYRIGHT

TRADEMARKS

MELISSA DATA CORPORATION
22382 Avenida Empresa
Rancho Santa Margarita, CA 92688

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

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

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

Document Code: DQSQSN
Revision Number: 100624.035
Last update: June 24, 2010

WELCOME TO THE DATA QUALITY SUITE

These customizable developer tools have been designed to help you efficiently manage your contact data for superior performance and profitablity. 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 and update area codes, parse and genderize names, validate email addresses, and more.

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's functionality is divided into four interfaces: AddressCheck, Parse, StreetData, and ZipData.

- **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 rual route to a standard city-street address) and automatically updates the address.

  The CASS Certified™ address checking logic includes DPV and LACS^{Link} processing mandatory to qualify your mailings for postal discount 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 mailing list by providing additional information, such as Congressional District, County name and FIPS code, time zone and more.

- 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.

- 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 Ave" is not a valid address for the ZIP Code, the StreetData interface can determine that the number 123 falls within a valid range for "Main St," allowing you to use 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 geographic data about one or more city names that match the input ZIP Code, 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 with the state. This can be useful to help correct misspelled city names in your mailing list.

Add-ons to Address Object

- Canadian Address Object — Increase the accuracy of your Canadian addresses. Use this API with your custom PC & Web applications to validate, correct and standardize Canadian addresses and qualify your mailings to meet Canada Post requirements.
- 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 method for shipping parcels by identifying residential addresses... therefore maximizing your savings potential.

About Phone Object

Use Phone Object to reduce data entry errors, catch area code splits early, save time looking up area codes, provide dealer lookup for customers, and link U.S. numbers to city, state, ZIP Code, and county.

Phone Object takes a phone number with an area code, parses it and retrieves geographic data corresponding to the area code and prefix. It can also update phone numbers that have undergone a recent area code split, helping to keep your data current and error-free.

Also, by parsing your phone number data and relating each area code and prefix to geographic information, such as longitude and latitude, Phone Object can provide an additional tool for planning a marketing campaign.

About Name Object

Use Name Object to increase response rates with personalized messages, merge individual components into form documents, identify gender makeup of your list for more targeted marketing, and reduce waste on fraudulent entries.

Name Object takes a string containing one or two full names, such as "John James Smith" or "Mr. John J. Smith and Ms. Mary Jones," and breaks it into first, middle and last names, as well as any titles, generational suffixes, or other information.

Name Object will also attempt to indicate the gender of the person based on the first name, if possible. For instance, "James" would return an "M" for male, "Cheryl" an "F" for female but names like "Chris" or "Tracy" can be either male or female and would return "N" for "neutral." Name Object can be configured to give preference to either male or female for lists that have a strong bias toward one gender or another.

Name Object will also flag names containing possible vulgar words to help you screen out possible hoaxes, pranks, nuisance names ("Mickey Mouse") and possible Company Names ("Microsoft, Inc"). It can also create salutations, such as "Dear Mr. Jones," using the parsed name information (Salutation uses the first person's name if two full names are present).

About Email Object

Email Object gives integrators a simple way to verify, correct, and standardize email address domains, to ensure legitimate email addresses are captured in contact databases. Email Object can be used in batch or at the point of entry, and has three optional levels of verification: Syntax, Local Database, and MXlookup. Email Object also allows for easy customization of domain changes, common misspellings, and suppression of bad domains.

Features Include:

- Check for and remove syntax errors: (@@#$5!..`]>
- Standardize Casing bud@800MAIL.COM -> bud@800mail.com
- Check for and update domain name changes @home.com –> cox.net
- Check for and correct top level domain name (TLD) .con-> .com
- Check for and correct misspellings in the domain: yohoo.com -> yahoo.com
- Validate a domain name against a database of known good and bad domains
- Validate domains via an optional MaileXchange (MX) Lookup
- Parse email addresses into various components: bud, @, 800mail, com
- Return the top level domain description

An Example

The following is an example of what the Data Quality Suite can return from a given set of data

Contact Data Submitted

Name ..... Mr. John Wayne Brown, Jr & Mrs. Mary Elizabeth Brown
Company ..... Melissa Data
Address ..... 22382 Emprisa
ZIP Code 92688
Phone 7145895200x111
Email ..... JOHN@800Mail.coom

Address Data Returned

Company ..... Melissa Data
Address ..... 22382 Avenida Empresa
City ..... Rancho Santa Margarita
City Abbreviation ..... Rcho Sta Marg
State ..... CA
ZIP 92688
Plus4 2112
Suite ..... [N/A]
Carrier Route C056
Delivery Point 821
County Name ..... Orange
County FIPS C06059
Time Zone ..... Pacific
PMSA 5945
LACS ..... [N/A]
Address Type ..... S
Private Mailbox ..... [N/A]
Latitude 33.6480
Longitude ..... -117.6000
Congressial Dist 48
Status ..... Verified

Name Data Returned

FIrst Name ..... John
Middle Name ..... Wayne
Last Name ..... Brown
Prefix ..... Mr
Suffix ..... Jr
Gender ..... Male
First Name 2 ..... Mary
Middle Name 2 ..... Elizabeth
Last Name 2 ..... Brown
Prefix 2 ..... Mrs
Gender 2 ..... Female

Phone Data Returned

Area Code 714
Prefix 589
Suffix 5200
Extension 111
Distance ..... [N/A]
New Area Code 949
Status ..... Corrected

Email Data Returned

TLD ..... com
domain ..... melissadata
TLD Description ..... Operated by Verisign
Email ..... john@melissadata.com
Status ..... Verified

Trial Versions

All of the Data Quality Tools found on the DVD Suite can be used in demo mode to help you get familiar with our products before purchasing a license. The trial versions contain the same features found in the full versions, but are limited. Address Object and GeoCoder Object will only process ZIP™ codes in Nevada. Phone Object will only process Nevada area codes.

The trial versions for Name Object and Email Object can be used with a free 30-day license, after which the object must be purchased to continue using.

If you wish to purchase any or all of the tools in this Suite, simply contact your sales representative at 1-800-MELISSA (1-800-635-4772). Upon purchasing the product, you will be provided with a license string to unluck its full functionality.

WHERE TO FIND HELP

Reference Guide

The Data Quality Suite Reference Guide contains the functions available for the various objects. A PDF file of this guide is located on your DVD-ROM.

To view using other operating systems, you must have Adobe Acrobat Reader installed on your computer. This program can be downloaded from the Adobe web site (http://www.adobe.com).

The Quick Start Guide and the Reference Guide are also available in HTML format. You can access this from the Technical Support web page on the Melissa Data web site (www.MelissaData.com).

Melissa Data Web Site

Check out the technical support section on our web site at www.MelissaData.com, where you can view the current application notes and FAQs. Our web site also contains the latest product information for the Data Quality Suite and other Melissa Data products.

Call Us Toll Free

If you need help using the Data Quality Suite, please call Technical Support toll free at 1-800-MELISSA (1-800-635-4772). Our technical support staff is available Monday through Friday from 6 a.m. - 5 p.m. Pacific Standard Time. You can also post a question on our forums at http://forum.MelissaData.com or send an e-mail to Tech@MelissaData.com.

SYSTEM REQUIREMENTS

The Data Quality Suite is shipped on a DVD. It is recommended that you copy the data files to your local or network hard drive in order to access the data faster. Because these objects are in essence programmer's tools, they should be installed on a system that has a development environment in order to utilize the power of the objects.

The following are additional hardware and software requirements:

- 2GB hard disk space (for data files)
- Windows 2000, XP, Server 2003 or Server 2008 (32 or 64 bit); Pentium III or better. Most Windows-based programming languages.

The actual deployment system does not require use of the development tools.

INSTALLING THE DATA QUALITY SUITE

1. Close all applications that are open, including any anti-virus and e-mail software.
2. Place the Data Quality Suite DVD in your DVD-ROM drive.
3. Click the **Start** button, and then select **Run** from the pop-up menu. The *Run* dialog box opens.
4. Type your DVD drive letter followed by `:\SETUP.EXE` in the *Command Line* box. (For example, `D:\SETUP.EXE`). After typing the path, click **OK**.
5. Read the license agreement, and then click **Yes** if you agree to the terms. (To proceed with the installation, you must accept this agreement.)
6. Select the component or components that you want to install by placing a check mark next to its name and clicking **Next**.
7. The Data Quality Suite
8. Click **Next**.
9. Click **Install**.

FILE LOCATIONS

Most of the files are placed in subdirectories of `C:\Program Files\Melissa Data\DQT\`. During the installation process, there may be some files placed in your Windows system directory — these files are required by the Microsoft Visual C++ runtime.

CONFIGURING THE DATA QUALITY SUITE

A current license string is required to use the full functionality of the object in the Data Quality Suite. Without a license string, they will work in demo mode.

The license string should be entered as an environment variable. This allows you to update your license string without editing and recompiling your code. You must still call the SetLicenseString function, but you simply pass an empty string as the parameter.

These are the environment variable names for the objects in the Data Quality Suite:

- **Address:** `MDADDR_LICENSE`
- **Phone:** `MDPHONE_LICENSE`
- **Name:** `MDNAME_LICENSE`
- **Email:** `MDEMAIL_LICENSE`

Setting the License String Environment Variable

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 the variable name (MDADDR_LICENSE, MDPHONE_LICENSE, MDNAME_LICENSE, or MDEMAIL_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. You may need to quit and restart the development environment to incorporate the changes.

Sample Implementation

This section describes how to build simple applications using the objects in the Data Quality Suite. We will be using Visual Basic.NET for illustration purposes.

Instantiation and Initialization

Declaring and creating an instance of the object being used is followed by one or more initialization steps to prepare it for use. These steps only need to be performed once per instance of the object in use.

Lookup

In this section, the required data is passed to the input functions and the `VerifyAddress` function to process the data is called. Many functions in this suite accept the input data as parameters and do not require that any "set" functions be called before calling them.

Processing

In the processing stage, you'll display or examine the return values of the "get" functions.

Termination

This stage involves destroying the current instance of the object in use when you close the application, freeing up memory to be used by other processes.

Address Object Sample — AddressCheck

In order to use the .NET wrapper, you will need to copy two DLL files from the Data Quality Suite DVD to a local folder. These files, mdAddrNET.dll and mdAddrCS.dll are located in the `address\windows\interfaces\net` folder. Copy the files to a location inside your project folder.

The .NET wrapper must be able to find the Address Object's standard DLL file, mdAddr.dll. The simplest method is to copy this file from the installed location, normally "C:\Program Files\Melissa DATA\DQT\AddrObj," to the same folder that contains mdAddrNET.dll and mdAddrCS.dll.

Step 1 — Instantiation: Add a Project Reference

Every Visual Basic project that uses an external component like Address Object has to start with a project reference.

1. Select *Project* > *Add Reference...*
2. Click on the Browse tab of the Reference dialog and open the folder where you copied the two .NET wrapper files.
3. Select the file named *mdAddrNET.dll*.
4. Click **OK**.

Step 2 — Instantiation: Add an Object Declaration

Begin by declaring the object as a new object variable.

VB.NET

```
Private addrObj As New MelissaData.mdAddr
```

C#.NET

```
mdAddr addrObj = new MelissaData.mdAddr
```

Step 3 — Initialization: Set the License String

Call the SetLicenseString function for Address Object to retrieve the license string from the MDADDR_LICENSE environment variable.

VB.NET

```
addrObj.SetLicenseString("")
```

C#.NET

```
addrObj.SetLicenseString("");
```

Step 4 — Initialization: Error Checking

The next step is to tell Address Object where it can find its data files. The initialization will either be successful, or an error message will display stating why the error occurred.

The path to the Canadian data files is only necessary if you are using those options. The path to the U.S. data files can be omitted if you are only processing Canadian address data.

The initialization error code is returned as a text string.

VB.NET

```
Private dPath = "C:/Program Files/Melissa Data/DQT/Data Files"
addrObj.SetPathToUSFiles(dPath)
addrObj.SetPathToDPVDataFiles(dPath)
addrObj.SetPathToLACSLinkDataFiles(dPath)
addrObj.SetPathToCanadaFiles(dPath)

If (addrObj.InitializeDataFiles() <> 0) Then
    InitErrorString = addrObj.GetInitializeErrorString
Else
    InitErrorString = addrObj.GetInitializeErrorString
    DatabaseDate = addrObj.GetDatabaseDate
    ExpDate = addrObj.GetExpirationDate
    BuildNum = addrObj.GetBuildNumber
End If
```

C#.NET

```
private static string dPath = "C:\\Program Files\\Melissa
       Data\\DQT\\Data Files";
addrObj.SetPathToUSFiles(dPath);
addrObj.SetPathToDPVDataFiles(dPath);
addrObj.SetPathToLACSLinkDataFiles(dPath);
addrObj.SetPathToCanadaFiles(dPath);

If (addrObj.InitializeDataFiles() != 0) {
    InitErrorString = addrObj.GetInitializeErrorString();
} Else {
    InitErrorString = addrObj.GetInitializeErrorString();
    DatabaseDate = addrObj.GetDatabaseDate();
    ExpDate = addrObj.GetExpirationDate();
    BuildNum = addrObj.GetBuildNumber();
}
```

Step 5 — Lookup: Input Required for VerifyAddress Function

The minimum required input is a street address, plus either the city and state or the five-digit ZIP Code™. Company name may be necessary for an accurate ZIP + 4® in the case of business or other facility addresses that have their own unique nine-digit ZIP Code.

VB.NET

```
addrObj.SetAddress("22382 Avenida Empresa")
addrObj.SetZip("92688")
addrObj.SetCompany("Melissa Data Corp")
```

C#.NET

```
addrObj.SetAddress("22382 Avenida Empresa");
addrObj.SetZip("92688");
addrObj.SetCompany("Melissa Data Corp");
```

Step 6 — Lookup: Calling the VerifyAddress Function

If the return value is false, then the address could not be properly coded.

VB.NET

```
Private Result as Boolean
Result = addrObj.VerifyAddress
```

C#.NET

```
private bool Result = false;
result = addrObj.VerifyAddress();
```

Step 7 — Processing: Checking the Results

If the `VerifyAddress` could not code the input address, you can check the reason by querying the return value of the GetResults function.

VB.NET

```
ResultCode = addrObj.GetResults
```

C#.NET

```
private string resultCode = addrObj.GetResults();
```

The `GetResults` function replaces the `GetStatusCode`, `GetErrorCode`, `GetErrorString`, `GetSuiteStatus` and `GetDPVFootnotes` functions. Those functions should be considered deprecated going foward, but if your existing code requires that you continue to use these functions, `GetStatusCode` should be called first. If Address Object could not return a result, then the `GetErrorCode` function will give you additional information about the reason why the query failed. Both functions return text strings that are exactly one character long. `GetErrorString` returns a fuller description of the results returned by `GetErrorCode`.

Step 8 — Processing: Retrieving the Results

The `VerifyAddress` function populates return values of numerous functions, a few of which are illustrated in the example below. For a complete list of the values returned by the `VerifyAddress` function, see the Address Object Reference Guide.

VB.NET

```
Address1 = addrObj.GetAddress
Address2 = addrObj.GetAddress2
Suite = addrObj.GetSuite
```

```
City = addrObj.GetCity
State = addrObj.GetState
Zip = addrObj.GetZip
Plus4 = addrObj.GetPlus4
CountyName = addrObj.GetCountyName
```

C#.NET

```
private string Address1 = addrObj.GetAddress();
private string Address2 = addrObj.GetAddress2();
private string Suite = addrObj.GetSuite();
private string City = addrObj.GetCity();
private string State = addrObj.GetState();
private string Zip = addrObj.GetZip();
private string Plus4 = addrObj.GetPlus4();
private string CountyName = addrObj.GetCountyName();
```

These are just the basics. See the Data Quality Suite Reference Guide for information about all the information that Address Object can return.

Step 9 — Termination: Destroying the Instance

Be sure to recycle your memory bits. We do not believe Address Object leaks allocated resources, however, you must properly destroy Address Object to force the release of memory.

VB.NET

```
addrObj.Dispose()
```

C#.NET

```
addrObj.Dispose();
```

Address Object Sample — Street Data Interface

The StreetData interface returns a list of all possible address ranges matching a specific street name or portion of a street name within a given ZIP Code. This is useful when VerifyAddress returns an error code for an address indicating that the address falls outside any known deliverable range. Rather than discarding the address as useless, you can attempt to match it against known ranges on the same street or similarly named streets in the same ZIP Code. This might give you a list of alternate possible addresses.

If you are using Address Object as part of your web application and a customer enters a bad address, or their address matches more than one record, StreetData enables you to catch it immediately and present the customer with the opportunity to select the correct address.

The FindStreet function accepts a full or partial street name ("MA*" for all streets beginning with the letters M-A) and a five-digit ZIP Code, like this:

```
Result = StreetData.FindStreet("Ash*", "92899")
```

This function would then return values filled with the first address range that matches that street name and ZIP Code, such as Ash Street and Ashland Avenue. The information returned includes the street name, the high and low values of the street number range, the high and low

values of the Plus 4 portion of the ZIP Code, and the high and low values for the range of suite numbers, if any. FindStreet also returns indicators that show if the above numbers in the range are odd, even or both.

You could then use the IsAddressInRange2 function to determine if a specific address falls within that range. If not, you would then call the FindStreetNext function to return the next matching range, and compare the address to that range, repeating until you exhaust all possibilities.

Step 1 — Instantiation: Declaring and Creating an Instance

See Step 1 above under the AddressCheck sample for information on adding Address Object to your project.

VB.NET

```
Dim StreetData As New MelissaData.mdStreet
```

C#.NET

```
private mdStreet StreetData = New MelissaData.mdStreet;
```

Step 2 — Initialization: Set the License String

Follow the same process used in Step 3 under the AddressCheck interface above.

Step 3 — Initialization: Initialize the Data Files

StreetData uses a different initialization method than AddressCheck. The Initialize function calls for two string values containing the path to the Address Object's data files named mdAddr.dat, mdAddr.nat, mdAddr.lic, and mdAddr.str.

There is a third, optional parameter that is no longer used. If your programming environment does not support optional parameters, pass an empty string to this function as a third parameter.

VB.NET

```
Private Result As AddressObjectErrorCodes
Result = StreetData.Initialize(DataPath, NationalPath)
If Result <> 0 Then
   Msgbox (StreetData.GetInitializeErrorString)
End If
```

C#.NET

```
AddressObjectErrorCodes Result = new AddressObjectErrorCodes;
Result = StreetData.Initialize(DataPath, NationalPath);
If Result != 0 {
   Console.WriteLine (StreetData.GetInitializeErrorString());
}
```

Step 4 — Lookup: Calling the FindStreet Function

The FindStreet function returns the first address range that matches the street name or partial name and ZIP Code passed as parameters.

VB.NET

```
Private Result as Boolean
Result = StreetData.FindStreet(StreetName, ZipCode)
```

C#.NET

```
bool Result = false;
Result = StreetData.FindStreet(StreetName, ZipCode);
```

Step 5 — Processing: Calling the IsAddressInRange2 Function

If the FindStreet function finds a range which matches the street name and ZIP Code submitted, it returns values to several functions which can then be compared to specific address data to see if the address falls within the returned range.

The IsAddressInRange2 function compares a submitted value to a high and low value to see if the value falls within the range. The function can also exclude odd or even values. You can use this function as a backstop for VerifyAddress to check an address that could not be verified to find alternate possible addresses that might be valid.

VB.NET

```
Private InRange As Boolean, High as String, Low As String, Odd As
       String

Do While Result
    High = StreetData.GetPrimaryRangeHigh
    Low = StreetData.GetPrimaryRangeLow
    Odd = StreetData.GetPrimaryRangeOddEven
    InRange = StreetData.IsAddressInRange2(Range, High, Low, Odd)
    If InRange Then
       Msgbox "Match found"
    End If
```

C#.NET

```
bool InRange = false;
string High = "";
string Low = "";
string Odd = "";

while Result {
    High = StreetData.GetPrimaryRangeHigh();
    Low = StreetData.GetPrimaryRangeLow();
    Odd = StreetData.GetPrimaryRangeOddEven();
    InRange = StreetData.IsAddressInRange2(Range, High, Low, Odd);
    if InRange {
       Console.WriteLine("Match found");
    }
```

Step 6 — Lookup: Find the Next Match with FindStreetNext

The FindStreetNext function returns the next match to the pattern passed to the last call to the FindStreet function. If there are no more matches, the function returns a value of false. This allows you to loop through all possible street data matches and use the IsAddressInRange2 function to find any that might match your address. The complete structure might look something like the code below.

VB.NET

```
        Result = StreetData.FindStreetNext
    Loop
```

C#.NET

```
        Result = StreetData.FindStreetNext();
    }
```

Step 7 — Termination: Destroying the Instance

Be sure to recycle your memory bits. We do not believe Address Object leaks allocated resources, however, you must properly destroy your StreetData Object to force the release of memory.

VB.NET

```
StreetData.Dispose()
```

C#.NET

```
StreetData.Dispose();
```

Address Object Sample — ZipData Interface

ZIP Data interface returns both ZIP Code data based on city and state information and city/state information based on ZIP Code. Using the example of a web application processing customer addresses, this interface could be used in the event that the customer enters an unrecognized city, state or ZIP Code. After the VerifyAddress function catches errors, the ZipData functions can generate a list of cities or ZIP codes to be presented to the customers as alternatives to what they entered.

Step 1 — Instantiation: Declaring and Creating an Instance

See Step 1 under the Address Check sample above for information on adding Address Object to your project.

VB.NET

```
Dim ZipData As New MelissaData.mdZip
```

C#.NET

```
mdZip ZipData = New MelissaData.mdZip
```

Step 2 — Initialization: Set the License String

Follow the same process used in Step 3 under the AddressCheck interface above.

Step 3 — Initialization: Initialize the Data Files

ZipData uses the same initialization functions as the Street Data interface. See Step 3 under StreetData for an example.

Step 4 — Lookup: Calling the FindZip Function

The FindZip function retrieves the first record found for the submitted five-digit ZIP Code, which includes city name, state, county information and geographical information, and populates the return value of the corresponding functions. Often there will be only one unless the ZIP Code covers more than one city or there is at least one alternative name for a city. If the record returned includes the official preferred city name, a function called GetLastLineIndicator will return the letter "L." You can use this function to standardize your address data by using only the official city name.

The FindZipNext function retrieves the next record for the last call to FindZip, if any. If there is another record, this function returns true, false otherwise.

VB.NET

```
Private Result As Boolean
Result = ZipData.FindZip("92688")
While Result
    AreaCode = ZipData.GetAreaCode
    Automation = ZipData.GetAutomation
    City = ZipData.GetCity
    State = ZipData.GetState
    Lat = ZipData.GetLatitude
    Long = ZipData.GetLongitude
    TZ = ZipData.GetTimeZone
    TZCode = ZipData.GetTimeZoneCode
    Result = ZipData.FindZipNext
Loop
```

C#.NET

```
bool Result = false;
Result = ZipData.FindZip("92688");
While (Result) {
    private string AreaCode = ZipData.GetAreaCode();
    private string Automation = ZipData.GetAutomation();
    private string City = ZipData.GetCity();
    private string State = ZipData.GetState();
    private string Latitude = ZipData.GetLatitude();
    private string Longitude = ZipData.GetLongitude();
    private string TZ = ZipData.GetTimeZone();
    private string TZCode = ZipData.GetTimeZoneCode();
    Result = ZipData.FindZipNext();
}
```

See the Data Quality Suite Reference Guide for a complete list of functions for retrieving the values returned by the different ZipData functions.

Step 5 — Lookup: Calling the FindZipInCity Function

The FindZipInCity functions similarly to FindZip, except that, along with FindZipInCityNext, it retrieves every five-digit ZIP Code associated with a given city and state, and populates the return values of the corresponding functions. This function allows you to expand your data by easily associating records found in the same city.

The FindZipInCityNext function retrieves the next record for the last call to FindZipInCity, if any. If there is another record, this function returns true, false otherwise.

VB.NET

```
Private Result As Boolean
Result = ZipData.FindZipInCity("Rancho Santa Margarita", "CA")
While Result
    Automation = ZipData.GetAutomation
    City = ZipData.GetCity
    State = ZipData.GetState
    Lat = ZipData.GetLatitude
    Long = ZipData.GetLongitude
    TZ = ZipData.GetTimeZone
    TZCode = ZipData.GetTimeZoneCode
    Result = ZipData.FindZipInCityNext
Loop
```

C#.NET

```
bool Result = false
Result = ZipData.FindZipInCity("Rancho Santa Margarita", "CA");
While (Result) {
    private string Automation = ZipData.GetAutomation();
    private string City = ZipData.GetCity();
    private string State = ZipData.GetState();
    private string Latitude = ZipData.GetLatitude();
    private string Longitude = ZipData.GetLongitude();
    private string TZ = ZipData.GetTimeZone();
    private string TZCode = ZipData.GetTimeZoneCode();
    Result = ZipData.FindZipInCityNext();
}
```

See the Data Quality Suite Reference Guide for a complete list of the different ZipData functions.

Step 6 — Lookup: Calling the FindCityInState Function

The FindCityInState function returns the first city name and state found that matches a partial or whole city name and state. For example, passing both "Los A*" and "CA" to this function would return "Los Angeles," "Los Alamitos" and "Los Altos," among others. You might use this with a web application if a customer enters an unrecogized city name. Pass the state and first letter or two of the city name to FindCityInState to present alternatives for the customer to select.

A successful call to each ZipData function populates the return values for a series of functions with the associated data. For example, the following code would retrieve the results for a call to the FindCitiesInState function.

The FindCityInStateNext function retrieves the next record for the last call to FindCityInState, if any. If there is another record, this function returns true, false otherwise.

VB.NET

```
Result = ZipData.FindCityInState("Rancho*", "CA")
While Result
    City = ZipData.GetCity
    Abbrev = ZipData.GetCityAbbreviation
    State = ZipData.GetState
    Result = ZipData.FindCityInStateNext
Loop
```

C#.NET

```
bool Result = ZipData.FindCityInState("Rancho*", "CA");
While (Result) {
    private string City = ZipData.GetCity();
    private string Abbrev = ZipData.GetCityAbbreviation();
    private string State = ZipData.GetState();
    private string Result = ZipData.FindCityInStateNext();
}
```

Step 7 — Termination: Destroying the Instance

Be sure to recycle your memory bits. We do not believe Address Object leaks allocated resources, however, you should properly destroy any instance of the ZipData interface to force the release of memory.

VB.NET

```
ZipData.Dispose()
```

C#.NET

```
ZipData.Dispose();
```

Address Object Sample — Parse Interface

Step 1 — Instantiation: Declaring and Creating an Instance

See Step 1 under the Address Check sample above for information on adding Address Object to your project.

VB.NET

```
Dim Parser As New MelissaData.mdParse
Private Result As Boolean
Private dPath As String
dPath = "C:/Program Files/Melissa Data/DQT/Data Files"
Result = Parser.Initialize(dPath)
```

C#.NET

```
mdParse Parser = New MelissaData.mdParse;
bool Result = false;
string dPath = "C:\\Program Files\\Melissa Data\\DQT\\Data
       Files";
Result = Parser.Initialize(dPath);
```

Step 2 — Lookup: Calling the Parse Function

The Parse function only requires you to pass a string value containing the address to be parsed as a parameter. This function will always attempt to parse the data passed to it and does not return any error code.

VB.NET

```
Parser.Parse(StreetAddress)
```

C#.NET

```
Parser.Parse(StreetAddress);
```

Step 3 — Processing

The Parse function then populates the return values for several functions with the separate parts of the parsed address. These are PostDirection, PreDirection, PrivateMailboxName, PrivateMailboxNumber, Range, StreetName, Suffix, SuiteName and SuiteNumber.

ParseNext retrieves an alternative parsing of the address passed to the most recent call to the Parse function. If no more possibilities are found, the function returns a false value, otherwise it returns a true and populates the return values of the same functions that the Parse function populates.

VB.NET

```
Private Proceed As Boolean
Proceed = True

Do While Proceed = True
    Range = Parser.GetRange
    PreDir = Parser.GetPreDirection
    StreetName = Parser.GetStreetName
    Suffix = Parser.GetSuffix
    PostDir = Parser.GetPostDirection
    SuiteName = Parser.GetSuiteName
    SuiteNumber = Parser.GetSuiteNumber
    PMBName = Parser.GetPrivateMailBoxName
    PMBNumber = Parser.GetPrivateMailBoxNumber

    Proceed = Parser.ParseNext
Loop
```

C#.NET

```
bool Proceed = true
```

```
while (Proceed) {
    Range = Parser.GetRange();
    private string PreDir = Parser.GetPreDirection();
    private string StreetName = Parser.GetStreetName();
    private string Suffix = Parser.GetSuffix();
    private string PostDir = Parser.GetPostDirection();
    private string SuiteName = Parser.GetSuiteName();
    private string SuiteNumber = Parser.GetSuiteNumber();
    private string PMBName = Parser.GetPrivateMailBoxName();
    private string PMBNumber = Parser.GetPrivateMailBoxNumber();

    Proceed = Parser.ParseNext();
}
```

Step 4 — Lookup: Parsing Last Line Information

The LastLineParse function will break up a single string containing city, state and ZIP Code information. This is useful if your address information comes to you as lines of text rather than individual values.

VB.NET

```
Private LastLine As String
LastLine = "Rancho Santa Margarita, CA 92688-2112"
Parser.LastLineParse(LastLine)
City = Parser.GetCity
State = Parser.GetState
Zip = Parser.GetZip
Plus4 = Parser.GetPlus4
```

C#.NET

```
private string LastLine = "Rancho Santa Margarita, CA 92688-
       2112";
Parser.LastLineParse(LastLine);
private string City = Parser.GetCity();
private string State = Parser.GetState();
private string Zip = Parser.GetZip();
private string Plus4 = Parser.GetPlus4();
```

Step 5 — Termination: Destroying the Instance

Be sure to recycle your memory bits. We do not believe Address Object leaks allocated resources, however, you must properly destroy Address Object to force the release of memory.

VB.NET

```
Parser.Dispose()
```

C#.NET

```
Parser.Dispose();
```

PHONE OBJECT SAMPLE

The Phone Object retrieves geographic information based on a submitted phone number and also parses out the area code and prefix from the phone number. This includes longitude, latitude, county, and demographic area information based on the centroid of the area code/prefix combination.

Being able to sort records by area code, telephone prefix or both gives you additional tools to get more value out of your contact data.

In order to use the .NET wrapper, you will need to copy two DLL files from the Data Quality Suite DVD to a local folder. These files, mdPhoneNET.dll and mdPhoneCS.dll are located in the `phone\windows\interfaces\net` folder. Copy the files to a location inside your project folder.

The .NET wrapper must be able to find the Phone Object's standard DLL file, mdPhone.dll. The simplest method is to copy this file from the installed location, normally "C:\Program Files\Melissa DATA\DQT\PhoneObj," to the same folder that contains mdPhoneNET.dll and mdPhoneCS.dll.

Step 1 — Instantiation: Declaring and Creating an Instance

Every Visual Basic project that uses an external .NET component has to start with a project reference. See the procedure for the adding a reference to Address Object. The Phone Object file is named *mdPhoneNET.dll*.

Step 2 — Initialization: Create an Instance of Phone Object

Begin by declaring the object as a new object variable.

VB.NET

```
Private phoneObject As New MelissaData.mdPhone
```

C#.NET

```
mdPhone phoneObject = New MelissaData.mdPhone
```

Step 3 — Initialization: Setting the License String

Call the SetLicenseString function for Phone Object to retrieve the license string from the MDPHONE_LICENSE environment variable.

VB.NET

```
phoneObj.SetLicenseString ("")
```

C#.NET

```
phoneObj.SetLicenseString ("");
```

Step 4 — Initialization: Initialize the Data Files

The Initialize function requires a string value containing the path to Phone Object's data files.

VB.NET

```
Private Result As DataBaseErr
Result = phoneObj.Initialize(DataPath)
```

```
If Result <> 0 Then
   Msgbox (phoneObj.GetInitializeErrorString)
End If
```

C#.NET

```
DataBaseErr Result = phoneObj.Initialize(DataPath);
If (Result != 0) {
   Console.WriteLine (phoneObj.GetInitializeErrorString);
}
```

Step 5 — Lookup: Calling the Lookup Function

The Lookup function accepts a string containing a phone number and populates the return values of several functions if it successfully locates the area code and prefix in the database.

VB.NET

```
Dim Result As Boolean
Result = phoneObj.Lookup("949-589-5200", "92688")
```

C#.NET

```
bool Result = false;
result = phoneObj.Lookup("949-589-5200", "92688");
```

If the Lookup function returns a false value, an error has occurred. Check the GetErrorCode and GetStatusCode functions to determine the cause.

Step 6 — Processing: Retrieve the Return Values

A successful call to the Lookup function populates the return values of the functions listed below. The functions return string values. If you need to perform any calculations, you will need to convert numbers to the correct data type.

VB.NET

```
AreaCode = phoneObj.GetAreaCode
Prefix = phoneObj.GetPrefix
Suffix = phoneObj.GetSuffix
Ext = phoneObj.GetExtension
City = phoneObj.GetCity
Country = phoneObj.GetCountryCode
Fips = phoneObj.GetCountyFips
County = phoneObj.GetCountyName
Lat = phoneObj.GetLatitude
Long = phoneObj.GetLongitude
State = phoneObj.GetState
TimeZone = phoneObj.GetTimeZone
TimeZoneCode = phoneObj.GetTimeZoneCode
```

C#.NET

```
private string AreaCode = phoneObj.GetAreaCode();
private string Prefix = phoneObj.GetPrefix();
private string Suffix = phoneObj.GetSuffix();
```

```
private string Ext = phoneObj.GetExtension();
private string City = phoneObj.GetCity();
private string Country = phoneObj.GetCountryCode();
private string Fips = phoneObj.GetCountyFips();
private string County = phoneObj.GetCountyName();
private string Latitude = phoneObj.GetLatitude();
private string Longitude = phoneObj.GetLongitude();
private string State = phoneObj.GetState();
private string TimeZone = phoneObj.GetTimeZone();
private string TimeZoneCode = phoneObj.GetTimeZoneCode();
```

Step 7 — Processing: Correcting Area Codes

Phone Object also includes a CorrectAreaCode function, which checks the submitted phone number against a database of recently updated area codes and retrieves the new area code. The function then populates the GetAreaCode and GetNewAreaCode functions. If there has been an area code split, the updated area code will be passed to the GetNewAreaCode function. If there has been no change, the return values of both functions will be the same.

VB.NET

```
Result = phoneObj.CorrectAreaCode("949-589-5200", "92688")
If Result Then
    AreaCode = phoneObj.GetAreaCode
    NewAreaCode = phoneObj.GetNewAreaCode
Else
    Results = phoneObj.GetResults
End If
```

C#.NET

```
bool Result = phoneObj.CorrectAreaCode("949-589-5200", "92688");
If (Result) {
    private string AreaCode = phoneObj.GetAreaCode();
    private string NewAreaCode = phoneObj.GetNewAreaCode();
} else {
    private string Results = phoneObj.GetResults();
}
```

Step 8 — Termination: Destroying the Instance

Be sure to recycle your memory bits. We do not believe Phone Object leaks allocated resources, however, you must properly destroy any instance of Phone Object to force the release of memory.

VB.NET

```
phoneObj.Dispose()
```

C#.NET

```
phoneObj.Dispose();
```

NAME OBJECT SAMPLE

Melissa Data's Name Object takes a person's full name and parses it into first, middle and last. It can also recognize titles and honorifics such as "Ms." or "Dr.", as well as generational and educational suffixes like "Jr." or "MD." Name Object can, in many cases, recognize the gender of the first name and can also flag names containing possible vulgarities, screening out hoaxes or other bad data. This object enables you to sort by last name if your data comes to you with the contact name contained in a single string value. The ability to "genderize" your name data allows you to more accurately target direct mail advertising even when the customer did not supply gender information.

In order to use the .NET wrapper, you will need to copy two DLL files from the Data Quality Suite DVD to a local folder. These files, mdNameNET.dll and mdNameCS.dll are located in the `name\windows\interfaces\net` folder. Copy the files to a location inside your project folder.

The .NET wrapper must be able to find the Name Object's standard DLL file, mdName.dll. The simplest method is to copy this file from the installed location, normally "C:\Program Files\Melissa DATA\DQT\NameObj," to the same folder that contains mdNameNET.dll and mdNameCS.dll.

Step 1 — Instantiation: Declaring and Creating an Instance

Every Visual Basic project that uses an external .NET component has to start with a project reference. See the procedure for adding a reference to Address Object. The Name Object file is named *mdNameNET.dll*.

Step 2 — Initialization: Create an Instance of Name Object

Begin by declaring the object as a new object variable.

VB.NET

```
Dim nameObj As New MelissaData.mdName
```

C#.NET

```
mdName nameObj = New MelissaData.mdName;
```

Step 3 — Initialization: Setting the License String

Call the SetLicenseString function for Name Object to retrieve the license string from the MDNAME_LICENSE environment variable.

VB.NET

```
nameObj.SetLicenseString ("")
```

C#.NET

```
nameObj.SetLicenseString ("");
```

Step 4 — Initialization: Initialize the Data Files

Before initializing the data files, you must call the SetPathToNameFiles function using a string containing the path to the mdName data files.

VB.NET

```
nameObj.SetPathToNameFiles(strDataPath)
Result = nameObj.InitializeDataFiles()
If Result <> 0 Then
    Msgbox nameObj.GetInitializeErrorString
End If
```

C#.NET

```
nameObj.SetPathToNameFiles(strDataPath);
Result = nameObj.InitializeDataFiles()
If (Result != 0) {
    Console.WriteLine(nameObj.GetInitializeErrorString);
}
```

Step 5 — Lookup: Set the Name Object Options

The Name Object has several optional settings that control how the object genderizes names, builds salutations, and handles misspelled first names. The following lines show the default settings. See the Name Object Reference Guide for detailed information on each setting.

VB.NET

```
nameObj.SetFirstNameSpellingCorrection(1)
nameObj.SetGenderPopulation(1)
nameObj.SetGenderAggression(1)
nameObj.AddSalutation(Formal)
nameObj.AddSalutation(FirstLast)
nameObj.AddSalutation(Informal)
nameObj.AddSalutation(Slug)
nameObj.SetSalutationPrefix("Dear")
nameObj.SetSalutationSuffix(";")
nameObj.SalutationSlug("Valued Customer")
nameObj.PrimaryNameHint(4)
```

C#.NET

```
nameObj.SetFirstNameSpellingCorrection(1);
nameObj.SetGenderPopulation(1);
nameObj.SetGenderAggression(1);
nameObj.AddSalutation(Formal);
nameObj.AddSalutation(FirstLast);
nameObj.AddSalutation(Informal);
nameObj.AddSalutation(Slug);
nameObj.SetSalutationPrefix("Dear");
nameObj.SetSalutationSuffix(";");
nameObj.SalutationSlug("Valued Customer");
nameObj.PrimaryNameHint(4);
```

Step 6 — Lookup: Calling the Parse Function

The Parse function accepts a single string value containing a full name. Ideally, this name should be formatted as first name, middle name, and last name. Name Object can handle last name first if the string is properly formatted and punctuated (last name, comma, first name and middle name) or if PrimaryNameHint is set to handle an inverse name.

VB.NET

```
nameObj.SetFullName("Mr. John J. Smith, Jr. and Mrs. Mary Q.
       Jones")
nameObj.Parse
```

C#.NET

```
nameObj.SetFullName("Mr. John J. Smith, Jr. and Mrs. Mary Q.
       Jones");
nameObj.Parse();
```

Step 7 — Processing: Retrieving the Parsed Name

A successful call to the Parse function populates the return values of the following functions: FirstName ("John"), MiddleName ("J"), LastName ("Smith"), Prefix ("Mr."), Suffix ("Jr") and Gender ("M"). If a second full name was present, Name Object also populates the values for a second set of functions: FirstName2 ("Mary"), MiddleName2 ("Q"), LastName2 ("Jones"), Prefix2 ("Mrs."), Suffix2 (empty in this case) and Gender2 ("F").

The object also sets the return value of the GetSalutation function using the first name found in the full name, if more than one was present ("Dear Mr. Smith;")

VB.NET

```
FirstName = nameObj.GetFirstName
MiddleName = nameObj.GetMiddleName
LastName = nameObj.GetLastName
Prefix = nameObj.GetPrefix
Suffix = nameObj.GetSuffix
Gender = nameObj.GetGender
FirstName2 = nameObj.GetFirstName2
MiddleName2 = nameObj.GetMiddleName2
LastName2 = nameObj.GetLastName2
Prefix2 = nameObj.GetPrefix2
Suffix2 = nameObj.GetSuffix2
Gender2 = nameObj.GetGender2
Salutation = nameObj.GetSalutation
```

C#.NET

```
private string FirstName = nameObj.GetFirstName();
private string MiddleName = nameObj.GetMiddleName();
private string LastName = nameObj.GetLastName();
private string Prefix = nameObj.GetPrefix();
private string Suffix = nameObj.GetSuffix();
private string Gender = nameObj.GetGender();
```

```
private string FirstName2 = nameObj.GetFirstName2();
private string MiddleName2 = nameObj.GetMiddleName2();
private string LastName2 = nameObj.GetLastName2();
private string Prefix2 = nameObj.GetPrefix2();
private string Suffix2 = nameObj.GetSuffix2();
private string Gender2 = nameObj.GetGender2();
private string Salutation = nameObj.GetSalutation();
```

Step 8 — Processing: Check The Results

Check the output of the GetResults function to determine the degree of success and any problems with Parsing or Genderizing.

If First Name Spelling Correction was enabled, the results codes will also indicate if one or both first names were corrected.

VB.NET

```
Private ResultsOut As String, Result As String
Result = nameObj.GetResults
ResultsOut = ""
If (Result.Contains("NS01")) Then ResultsOut += "NS01: There were
      no errors. \r\n"
If (Result.Contains("NS03")) Then ResultsOut += "NS03: The
      spelling of the FirstName field was corrected. \r\n"
If (Result.Contains("NS04")) Then ResultsOut += "NS04: The
      spelling of the FirstName2 field was corrected. \r\n"
If (Result.Contains("NS02")) Then ResultsOut += "NS02: A Parse
      Error was found.... \r\n"
If (Result.Contains("NE01")) Then ResultsOut += "NE01: Two names
      were detected but the FullName string was not in a
      recognized format \r\n"
If (Result.Contains("NE02")) Then ResultsOut += "NE02: Multiple
      first names - could not accurately genderize \r\n"
If (Result.Contains("NE03")) Then ResultsOut += "NE03: A
      vulgarity was detected in the name \r\n"
If (Result.Contains("NE04")) Then ResultsOut += "NE04: A nuisance
      name [such as Mickey Mouse] was detected in the name \r\n"
If (Result.Contains("NE05")) Then ResultsOut += "NE05: The name
      contained words normally found in a company name \r\n"
If (Result.Contains("NE06")) Then ResultsOut += "NE06: The name
      contained a non-alphabetic character \r\n"
Print ResultsOut
```

C#.NET

```
string Result = nameObj.GetResults();
string ResultsOut = "";
If (Result.Contains("NS01")) ResultsOut += "NS01: There were no
      errors. \r\n";
```

```
If (Result.Contains("NS03")) ResultsOut += "NS03: The spelling of
      the FirstName field was corrected. \r\n";
If (Result.Contains("NS04")) ResultsOut += "NS04: The spelling of
      the FirstName2 field was corrected. \r\n";
If (Result.Contains("NS02")) ResultsOut += "NS02: A Parse Error
      was found.... \r\n";
If (Result.Contains("NE01")) ResultsOut += "NE01: Two names were
      detected but the FullName string was not in a recognized
      format \r\n";
If (Result.Contains("NE02")) ResultsOut += "NE02: Multiple first
      names - could not accurately genderize \r\n";
If (Result.Contains("NE03")) ResultsOut += "NE03: A vulgarity was
      detected in the name \r\n";
If (Result.Contains("NE04")) ResultsOut += "NE04: A nuisance name
      [such as Mickey Mouse] was detected in the name \r\n";
If (Result.Contains("NE05")) ResultsOut += "NE05: The name
      contained words normally found in a company name \r\n";
If (Result.Contains("NE06")) ResultsOut += "NE06: The name
      contained a non-alphabetic character \r\n";
Console.WriteLine(ResultsOut);
```

Step 9 — Termination: Destroying the Instance

Be sure to recycle your memory bits. We do not believe Name Object leaks allocated resources, however, you must properly destroy Name Object to force the release of memory.

VB.NET

```
nameObj.Dispose()
```

C#.NET

```
nameObj.Dispose();
```

EMAIL OBJECT SAMPLE

In order to use the .NET wrapper, you will need to copy two DLL files from the Data Quality Suite DVD to a local folder. These files, mdNameNET.dll and mdEmailCS.dll are located in the `name\windows\interfaces\net` folder. Copy the files to a location inside your project folder.

The .NET wrapper must be able to find the Name Object's standard DLL file, mdEmail.dll. The simplest method is to copy this file from the installed location, normally "C:\Program Files\Melissa DATA\DQT\EmailObj," to the same folder that contains mdEmailNET.dll and mdEmailCS.dll.

Step 1 — Instantiation: Declaring and Creating an Instance

Every Visual Basic project that uses an external .NET component has to start with a project reference. See the procedure for the adding a reference to Address Object. The Email Object file is named *mdEmaileNET.dll*.

Step 2 — Initialization: Create an Instance of Email Object

Begin by declaring the object as a new object variable.

VB.NET

```
Dim emailObj as New MelissaData.mdEmail
```

C#.NET

```
mdEmail emailObj = New MelissaData.mdEmail;
```

Step 3 — Initialization: Setting the License String and Data Path

Call the SetLicenseString function for Email Object to retrieve the license string from the MDEMAIL_LICENSE environment variable.

VB.NET

```
emailObj.SetLicenseString("")
```

C#.NET

```
emailObj.SetLicenseString("");
```

Step 4 — Initialization: Initializing the Data Files

Initialializing connects Email Object to its data files. If it does not initialize, then display the GetInitializeErrorString for an explanation of the error.

VB.NET

```
emailObj.SetPathToEmailFiles(DataPath)
If (emailObj.InitializeDataFiles() <> 0) Then
   MsgBox emailObj.GetInitializeErrorString()
End If
```

C#.NET

```
emailObj.SetPathToEmailFiles(DataPath);
If (emailObj.InitializeDataFiles() != 0) {
   Console.WriteLine(emailObj.GetInitializeErrorString());
}
```

Step 5 — Initialization: Setting the Verification Options

Email Object has five boolean functions to enable or disable certain options before verifying an email address. For more information on these functions, see the Data Quality Suite Reference Guide.

VB.NET

```
emailObj.SetCorrectSyntax(True)
emailObj.SetDatabaseLookup(True)
emailObj.SetMXLookup(True)
```

```
emailObj.SetStandardizeCasing(True)
emailObj.SetUpdateDomain(True)
```

C#.NET

```
emailObj.SetCorrectSyntax(True);
emailObj.SetDatabaseLookup(True);
emailObj.SetMXLookup(True);
emailObj.SetStandardizeCasing(True);
emailObj.SetUpdateDomain(True);
```

Step 6 — Processing: Verifying the Email Address

Verifying the email address is simply a matter of passing the email address to the VerifyEmail function, which returns a boolean true if successful. You can then extract the various parts of the standardized address using the functions shown below.

The GetChangeCode function returns a number that can be used to check what has been changed. Use the AND operator to check the first four bits.

If the VerifyEmail function cannot verify the submitted email address, Email Object returns the Error and Status Codes to reveal the reason why the address could not be successfully verified.

VB.NET

```
If emailObj.VerifyEmail(InputEmail) = True Then
    Mailbox = emailObj.GetMailBoxName
    TLD = emailObj.GetTopLevelDomain
    TLDDesc = emailObj.GetTopLevelDomainDescription
    EmailAddress = emailObj.GetEmailAddress
    Domain = emailObj.GetDomainName
    Private ChangeList As String = ""
    Private ChangeCode As Integer
    ChangeCode = emailObj.GetChangeCode
    If ((ChangeCode And 1) = 1) Then
        ChangeList += "Syntax Changed" + vbNewLine
    End If
    If ((ChangeCode And 2) = 2) Then
        ChangeList += "TLD Updated" + vbNewLine
    End If
    If ((ChangeCode And 4) = 4) Then
        ChangeList += "Domain Spelling" + vbNewLine
    End If
    If ((ChangeCode And 8) = 8) Then
        ChangeList += "Domain Updated" + vbNewLine
    End If
Else
    Status = emailObj.GetStatusCode
    ErrCode = emailObj.GetErrorCode
    ErrDesc = emailObj.GetErrorString
End If
```

C#.NET

```
If (emailObj.VerifyEmail(InputEmail)) {
    string Mailbox = emailObj.GetMailBoxName();
    string TLD = emailObj.GetTopLevelDomain();
    string TLDDesc = emailObj.GetTopLevelDomainDescription();
    string EmailAddress = emailObj.GetEmailAddress();
    string Domain = emailObj.GetDomainName();

    string ChangeList = "";
    int ChangeCode = emailObj.GetChangeCode;
    If ((ChangeCode And 1) = 1) {
        ChangeList += "Syntax Changed" + vbNewLine;
    }
    If ((ChangeCode And 2) = 2) {
        ChangeList += "TLD Updated" + vbNewLine;
    }
    If ((ChangeCode And 4) = 4) {
        ChangeList += "Domain Spelling" + vbNewLine;
    }
    If ((ChangeCode And 8) = 8) {
        ChangeList += "Domain Updated" + vbNewLine;
    }
} else {
    string Status = emailObj.GetStatusCode();
    string ErrCode = emailObj.GetErrorCode();
    string ErrDesc = emailObj.GetErrorString();
}
```

Step 7 — Termination: Destroying the Instance

Be sure to recycle your memory bits. We do not believe Email Object leaks allocated resources, however, you must properly destroy this instance of Email Object to force the release of memory.

VB.NET

```
emailObj.Dispose()
```

C#.NET

```
emailObj.Dispose();
```