Quick Start Guide

Data Quality Suite

Data Quality Suite

ADDRESS OBJECT

PHONE OBJECT

NAME OBJECT

EMAIL OBJECT

Quick Start Guide

COPYRIGHT

TRADEMARKS

CONTACT

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

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

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

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

Document Code: DQSQSG
Revision Number: 110816.049

Last update: August 16, 2011

Table of Contents

Thanks For Purchasing The Data Quality Suite

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 and update area codes, parse and genderize names, validate email addresses, and more.

About Address Object

Clean up contact data, removing bad or 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, 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 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 LACSLink® 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 and LACSLink 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 address data by providing additional information, such as Congressional District, County name and FIPS code, time zone, and several others.

  If Address Object is purchased with the Canadian Address option, the AddressCheck can be used to verify Canadian addresses.

- 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 separate parts. This is handy if your address data comes to you as lines of plain text instead of discrete fields.

  If Address Object is purchased with the Canadian Address option, the Parse interface can be used to parse Canadian addresses.

- 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.

If Address Object is purchased with the Canadian Address option, the StreetData interface can be used with Canadian addresses.

- 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.

  If Address Object is purchased with the Canadian Address option, the ZipData interface can be used with Canadian postal codes, cities, and provinces.

Add-ons to Address Object

- **Canadian Address 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 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.
- **AddressPlus** — AddressPlus goes beyond the Post Office's Suite$^{\text{Link®}}$ product, appending missing secondary address information to millions of business and residential addresses.
- **eLOT** — This feature appends enhanced Line-of-Travel numbers to addresses for improved presorting and greater postage discounts when used in conjunction with Melissa Data's Presort Object.

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 validates the number to either the 7-digit or 10-digit level. It can also identify phone numbers as land lines, wireless or Voice Over IP (VOIP) as well as distinguish between business and residential phone numbers. It can also add geographic data corresponding to the area code and prefix and 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 be used as a 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 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

Trial Versions

All of the Data Quality Tools found on the DVD 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 Phone Object will only process Nevada addresses and area codes, respectively.

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 unlock the full functionality.

Where to Find Help

Reference Guide

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

Many Linux versions come with a built-in reader, such as Xpdf, to view the Reference Guide.

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

Melissa Data Website

Check out the technical support section on our website at www.MelissaData.com, where you can view the current application notes and FAQs. Our website 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 tools in 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 - 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:

- 4GB hard disk space (3.5GB for Address Object, 50MB for Name Object, and 200MB for Phone Object)
- Microsoft® Windows® users — 64-bit Windows Vista, Windows 7, Server 2003 or Server 2008. Most Windows-based programming languages. .NET Framework 3.5 or better required to register the COM object.
- Linux users — Red Hat 64-bit (x64) distribution. GNU C++ 3.4 or later; glibc 3.2 or later.
- Solaris users — Solaris 8,9,10, SPARC platform, 32 or 64-bit. Sun Workshop *or* Sun ONE Studio compiler. G++ 3.3 and later can also be used.
- AIX users — Version 5.2 or 5.3; POWER, rs/6000, PPC, 64-bit. gcc 3.4.6 or Visual Age.
- HP-UX users — Version 11.11, 11.23; PA-RISC or Itanium, 64-bit. gcc 3.4.6 or aCC A.03.70.

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

Installing the Data Quality Suite

Windows

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 installer will display a screen showing the install location and the selected options. Click **Next** again.
8. Click **Install**.

Linux/Solaris/HP-UX/AIX

You do not need any special privileges when installing the objects in the Data Quality Suite, nor do you have to log in as `root`.

During installation, nothing will be modified outside of the target directory.

To install any of the objects:

1. Place the Data Quality Suite DVD in your DVD-ROM drive.
2. From the shell prompt (`$`), mount the DVD and run the applicable setup_main.sh script to install the object(s) to the desired directory.

All of the Unix-based OS sample programs assume that they can locate the required data files and object libraries in the current directory. It is not necessary to modify your `PATH` or `LD_LIBRARY_PATH`. If you prefer, you can run the sample program straight from the DVD-ROM — there is no need to install anything.

The final deployment install has to be done manually or by using your system administration utilities. Since the deployment standards vary widely, Melissa Data does not provide any specific instructions. Remember the following:

- The objects in the Data Quality Suite do not require any special privileges
- All files can be made read-only
- There is no need for a `setuid` or `setgid`, neither as file permissions nor anywhere in your application code

File Locations

Windows

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.

Uninstalling the Data Quality Suite

To remove any or all of the objects in the Data Quality Suite from your computer, do the following:

Windows:

1. Click the **Start** button.
2. Select **Settings** from the pop-up menu.
3. Click **Control Panel.**
4. Double-click **Add/Remove Programs**.

Linux/Solaris/HP-UX/AIX:

1. Verify that all files in the target directory can be safely deleted.
2. Type `rm -rf` *`target-directory`*.

   If the target directory contains files that cannot be deleted without impacting other software, you will need to manually erase only files from the Data Quality Suite.

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 named "MD_LICENSE." This allows you to update your license string without editing and recompiling your code.

The global MD_LICENSE environment variable replaces the old object-specific environment variables**,** `MDADDR_LICENSE`, `MDPHONE_LICENSE`, `MDNAME_LICENSE`, and `MDEMAIL_LICENSE`.

These environment variable names can still be used for existing installations, but new installations should use the MD_LICENSE environment variables.

Setting the License String Environment Variable

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 the variable name 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. You may need to quit and restart the development environment to incorporate the changes.

Linux/Solaris/HP-UX/AIX

Unix-based OS users can simply set their license string via the following:

```
    export MD_LICENSE=A1B2C3D4E5
```

If you decide to put this setting in your `.profile`, remember to restart your shell.

Sample Implementations

This section describes the logic behind simple applications using the objects in the Data Quality Suite. The samples are written in pseudocode so they can be easily adapted into almost any language.

Initialization

Declaring and creating an instance of the object being used is followed by the 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 input values are populated and the function to process the data is called. Some functions accept the input data as parameters and do not require that any other values be populated before calling them.

Processing

In the processing stage, you'll display or examine the return values of the various 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

Step 1 — Instantiation: Declare and Create an Instance

Begin by declaring the object as a new object variable.

```
    CREATE addObj As New Instance of AddressCheck
```

Step 2 — Initialization: Set the License String

You must call the SetLicenseString function for Address Object to retrieve the license string from the MDADDR_LICENSE environment variable.

```
    CALL SetLicenseString
```

Step 3 — 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.

```
    Set Result = New InitializeErrors
    CALL SetPathToUSFiles WITH DataPath
    CALL SetPathToDPVDataFiles WITH DataPath
    CALL SetPathToLACSLinkDataFiles WITH DataPath
    CALL InitializeDataFiles RETURNING Result
    CALL GetInitializeErrorString RETURNING InitError
```

When using the shared object in Linux or Solaris, use the target directory specified when the Data Quality Suite was installed.

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.

Step 4 — Initialization: Check the Database Date, Expiration Date and Build Number

The database behind Address Object cannot be used after its expiration date.

To check the database and expiration date:

```
    CALL GetDatabaseDate RETURNING DatabaseDate
    CALL GetExpirationDate RETURNING ExpirationDate
```

The build number is necessary for obtaining technical support from Melissa Data.

```
    CALL GetBuildNumber RETURNING BuildNumber
```

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.

```
    CALL SetAddress WITH "22382 Avenida Empresa"
    CALL SetZip WITH "92688"
    CALL SetCompany WITH "Melissa Data Corp"
```

Step 6 — Lookup: Calling the VerifyAddress function

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

```
SET Result = New Boolean
CALL VerifyAddress RETURING Result
```

Step 7 — Processing: Checking the Result Codes

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

Result codes have replaced the old status and error codes with a comma-delimited string of four-letter codes. These codes show the status of the last call to the VerifyAddress function, including the level of address matching, as well as the cause for any errors.

Result codes also include "change codes," which indicate any alterations made to address information during the standardization process.

```
CALL GetResults RETURNING Results
Process Results
```

Step 8 — Processing: Retrieving the Results

The VerifyAddress function populates the return values for various functions, a few of which are shown in the example below. For a complete list of the functions, see the Data Quality Suite Reference Guide.

- In Visual Basic, they are all declared as `Variants`. Therefore, Visual Basic will do an appropriate conversion implicitly.
- In C++, they are all declared as `const char *`. You will have to convert them explicitly if you need to use them in any calculations.

```
CALL GetAddress RETURNING Address1
CALL GetAddress2 RETURNING Address2
CALL GetSuite RETURNING Suite
CALL GetCity RETURNING City
CALL GetState RETURNING State
CALL GetZip RETURNING Zip5
CALL GetPlus4 RETURNING Plus4
CALL GetCountyName RETURNING CountyName
```

Step 9 — Processing: Finding Suggested Alternatives

If the submitted address could not be verified, use the FindSuggestion function to retrieve possible alternatives.

```
CALL FindSuggestion RETURNING SuggestionsFound
```

If the FindSuggestion function returns an integer value of 1, alternative addresses were found. Repeat the function calls in Step 7 to retrieve the suggested alternative.

If the function returns a zero value, no suggestion was found.

```
    CALL FindSuggestionNext RETURNING SuggestionsFound
```

If the FindSuggestionNext function returns an integer value of 1, more alternative addresses were found. Repeat the function calls in Step 7 to retrieve the suggested alternative.

If the function returns a zero value, no further suggestions were found.

Step 10 — Termination: Destroying the Instance

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

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.

```
    CREATE parsObj As New Instance of Parse
    CALL Initialize WITH DataPath RETURNING Result
```

Step 2 — Lookup: Calling the Parse Function

The Parse function only requires you to pass 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.

```
    CALL Parse WITH StreetAddress
```

If you are parsing a Canadian address, you would use the ParseCanadian function instead.

```
    CALL ParseCanadian WITH StreetAddress
```

Step 3 — Processing

The Parse function then populates the return values of several functions with string values containing the separate parts of the parsed address.

```
    CALL GetRange RETURNING Range
    CALL GetPreDirection RETURNING PreDirection
    CALL GetStreetName RETURNING StreetName
    CALL GetSuffix RETURNING Suffix
    CALL GetPostDirection RETURNING PostDirection
```

If you called the ParseCanadian function with a Canadian street address, you could also call the following functions:

```
    CALL GetRouteService RETURNING RouteService
    CALL GetLockBox RETURNING LockBox
```

```
CALL GetDeliveryInstallation RETURNING
      DeliveryInstallation
```

Step 4 — Lookup: Calling the ParseNext Function

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 Boolean true and populates the same return values as the Parse function.

```
WHILE ParseNext RETURNS True
    CALL GetRange RETURNING Range
    CALL GetPreDirection RETURNING PreDirection
    CALL GetStreetName RETURNING StreetName
    CALL GetSuffix RETURNING Suffix
    CALL GetPostDirection RETURNING PostDirection
ENDWHILE
```

If you originally called the ParseCanadian function with a Canadian street address, you could also call the following functions:

```
    CALL GetRouteService RETURNING RouteService
    CALL GetLockBox RETURNING LockBox
    CALL GetDeliveryInstallation RETURNING
      DeliveryInstallation
```

Step 5 — 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 input data comes to you as single lines rather than individual values.

```
CALL LastLineParse WITH LastLine
CALL GetCity RETURNING City
CALL GetState RETURNING State
CALL GetZip RETURNING Zip
CALL Plus4 = parsePtr.Plus4
```

Step 6 — Termination: Destroying the Instance

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

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 the VerifyAddress function of the AddressCheck Interface 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 “MA.”) and a five-digit ZIP Code.

This function would then populate the return values of several functions 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 Address Check sample for information on adding Address Object to your project.

```
Create streetObj As New Instance of StreetData
```

Step 2 — Initialization: Set the License String

You must call the SetLicenseString function for Address Object to retrieve the license string from the MD_LICENSE environment variable.

```
CALL SetLicenseString
```

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 and mdAddr.Nat.

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.

```
CALL Initialize WITH DataPath, NationalPath RETURNING
  Result As AddressObjectErrorCodes
If Result Is Not 0 THEN
```

```
        CALL GetInitializeErrorString RETURNING ErrorStr
        PRINT ErrorStr
    END IF
```

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.

```
    CALL FindStreet WITH StreetName, ZipCode RETURNING
      Result
```

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 that 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.

The Range value would be a single street number, possibly from a call to the Parse function of the Parse Interface.

```
    WHILE Result Is True
        CALL GetPrimaryRangeHigh RETURNING RangeHigh
        CALL GetPrimaryRangeLow RETURNING RangeLow
        CALL GetPrimaryRangeOddEven RETURNING OddEven
        CALL IsAddressInRange2 WITH (Range, RangeHigh,
            RangeLow, OddEven RETURNING InRange
        IF InRange Is True THEN
            PRINT "Match found"
        ENDIF
```

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.

```
        CALL FindStreetNext RETURNING Result
    Loop
```

Step 7 — Termination: Destroying the Instance

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

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.

```
Create zipObject As New Instance of ZipCodeData
```

Step 2 — Initialization: Set the License String

You must call the SetLicenseString function for Address Object to retrieve the license string from the MD_LICENSE environment variable.

```
CALL SetLicenseString
```

Step 3 — Intialization: Initialize the Data Files

ZipData uses the same initialization functions as the Street Data interface. See Step 2 under the StreetData Interface on page 15 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 values 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 value to standardize your address data by using only the official city name.

```
CALL FindZip WITH "92688" RETURNING Result
WHILE Result Is True
    CALL GetAreaCode RETURNING AreaCode
    //
    // See the Data Quality Suite Reference Guide for
    // a complete list of functions to return data
    // from call to FindZip.
```

```
        //
        CALL GetZipType RETURNING ZipType
```

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.

```
        CALL FindZipNext RETURNING Result
    ENDWHILE
```

Step 5 — Lookup: Calling the FindZipInCity Function

The FindZipInCity function 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.

```
    CALL FindZipInCity WITH "Rancho Santa Margarita",
      "CA" RETURNING Result
    WHILE Result Is True
        CALL GetAutomation RETURNING Automation
        //
        // See the Data Quality Suite Reference Guide for
        // a complete list of functions to return data
        // from call to FindZipInCity.
        //
        CALL GetZipType RETURNING ZipType
```

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.

```
        CALL FindZipInCityNext RETURNING Result
    ENDWHILE
```

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 unrecognized city name. Pass the state and first letter or two of the city name to FindCityInState to present alternatives for the customer to select.

```
    CALL FindCityInState WITH "Rancho*", "CA" RETURNING
      Result

    WHILE Result Is True
        CALL GetCity RETURNING City
        CALL GetState RETURNING State
```

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.

```
    CALL FindCityInStateNext RETURING Result
ENDWHILE
```

Step 7 — Termination: Destroying the Instance

Be sure to free allocated memory. 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.

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.

Step 1 — Instantiation: Declaring and Creating an Instance

Begin by declaring the object as a new object variable.

```
Create phoneObj As New Instance of PhoneCheck
```

Step 2 — Initialization: Setting the License String

You must call the SetLicenseString function for Phone Object to retrieve the license string from the MD_LICENSE environment variable.

```
CALL SetLicenseString
```

Step 3 — Initialization: Initialize the Data Files

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

```
CALL Initialize WITH DataPath RETURNING Result
IF Result Is True THEN
    CALL GetInitializeErrorString RETURNING ErrorStr
    PRINT ErrorStr
ENDIF
```

Step 4 — Lookup: Calling the Lookup Function

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

```
CALL Lookup WITH PhoneNumber, ZipCode Returning
  Result
```

The ZipCode, if included, enables Phone Object to geocode the phone number more accurately and populate the return value of the GetDistance function.

Step 5 — Processing: Checking the Result Codes

If the Lookup function returns a false value, an error has occurred. Check the GetResults function to determine the cause.

Result codes have replaced the old status and error codes with a comma-delimited string of codes. These codes show the status of the last call to the Lookup function, including the level of phone number match found and the cause for any errors.

```
CALL GetResults RETURNING Results
Process Results
```

Step 6 — Processing: Retrieve the Data

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.

```
CALL GetAreaCode RETURNING AreaCode
CALL GetPrefix RETURNING Prefix
CALL GetSuffix RETURNING Suffix
CALL GetExtension RETURNING Extension
CALL GetCity RETURNING City
CALL GetState RETURNING State
CALL GetLatitude RETURNING Latitude
CALL GetLongitude RETURNING Longitude
CALL Distance RETURNING Distance
```

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 return values of the GetAreaCode and GetNewAreaCode function. If there has been an area code split, the GetNewAreaCode function will return an updated area code. If there has been no change, both functions will return the same values.

```
CALL CorrectAreaCode WITH PhoneNumber, ZipCode
  RETURNING Result
IF Result Is True THEN
    CALL GetAreaCode RETURNING AreaCode
    CALL GetNewAreaCode RETURNING NewAreaCode
ENDIF
```

Step 8 — Termination: Destroying the Instance

Be sure to free allocated memory. 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.

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.

Step 1 — Instantiation: Declaring and Creating an Instance

Begin by declaring the object as a new object variable.

```
Create nameObj As New Instance of mdName
```

Step 2 — Initialization: Setting the License String

You must call the SetLicenseString function for Name Object to retrieve the license string from the MD_LICENSE environment variable.

```
CALL SetLicenseString
```

Step 3 — 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.

```
CALL SetPathToNameFiles WITH DataPath
CALL InitializeDataFiles Returning Result
IF Result Is True THEN
    CALL GetInitializeErrorString RETURNING ErrorStr
    PRINT ErrorStr
ENDIF
```

Step 4 — Lookup: Set the Name Object Options

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

```
CALL SetFirstNameSpellingCorrection WITH True
CALL SetGenderPopulation WITH Mixed
CALL SetGenderAggression WITH Neutral
CALL AddSalutation WITH Formal
CALL AddSalutation WITH FirstLast
CALL AddSalutation WITH Informal
CALL AddSalutation WITH Slug
CALL SetSalutationPrefix WITH "Dear"
CALL SetSalutationSuffix WITH ";"
CALL SetSalutationSlug WITH "Valued Customer"
CALL SetPrimaryNameHint WITH 4 Varying
```

Step 5 — 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 also handles all variations of this forward FullName pattern, and other Inverse name patterns if the PrimaryNameHint property is set accordingly.

```
CALL SetFullName WITH "Mr. John J. Smith, Jr. and
  Mrs. Mary Q. Jones"
CALL Parse
```

Step 6 — Processing: Checking the Result Codes

If the Parse function returns a false value, an error has occurred. Check the GetResults function to determine the cause.

Result codes have replaced the old status and error codes with a comma-delimited string of codes. These codes describe the status of the last call to Parse function, as well as the cause for any errors.

```
CALL GetResults RETURNING Results
Process Results
```

Step 7 — Processing: Retrieving the Parsed Name

A successful call to the Parse function populates the return values of the following functions: GetFirstName (“John”); GetMiddleName (“J”); GetLastName (“Smith”); GetPrefix (“Mr.”); GetSuffix (“Jr”); and GetGender (“M”). If a second name was present; Name Object also populates the return values of a second set of functions: GetFirstName2 (“Mary”); GetMiddleName2 (“Q”); GetLastName2 (“Jones”); GetPrefix2 (“Mrs.”); GetSuffix2 (empty in this case); and GetGender2 (“F”).

The object also populates the return value of the GetSalutation function using the name values found in the first full name of the Full Name string (“Dear Mr. Smith;”)

```
CALL GetFirstName RETURNING FirstName
CALL GetMiddleName RETURNING MiddleName
CALL GetLastName RETURNING LastName
CALL GetPrefix RETURNING Prefix
```

```
    CALL GetSuffix RETURNING Suffix
    CALL GetGender = namePtr.Gender
    CALL GetFirstName2 = namePtr.FirstName2
    CALL GetMiddleName2 = namePtr.MiddleName2
    CALL GetLastName2 = namePtr.LastName2
    CALL GetPrefix2 = namePtr.Prefix2
    CALL GetSuffix2 = namePtr.Suffix2
    CALL GetGender2 = namePtr.Gender2
    CALL GetSalutation = namePtr.Salutation
```

Step 8 — Termination: Destroying the Instance

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

Email Object Sample

Step 1 — Initialization: Declaring and Creating an Instance

Begin by declaring an instance of the Email Object.

```
    Create emailObj as New Instance of mdEmail
```

Step 2 — Initialization: Setting the License String and Data Path

You must call the SetLicenseString function for Email Object to retrieve the license string from the MD_LICENSE environment variable.

```
    CALL SetLicenseString
```

Step 3 — Initialization: Initializing the Data Files

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

```
    CALL SetPathToEmailFiles WITH DataPath
    CALL InitializeDataFiles RETURNING Result
    IF Result Is True THEN
        CALL GetInitializeErrorString RETURNING ErrorStr
        PRINT ErrorStr
    ENDIF
```

Step 4 — Initialization: Setting the Verification Options

Email Object has five 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.

```
    CALL SetCorrectSyntax WITH True
    CALL SetDatabaseLookup WITH True
    CALL SetMXLookup WITH True
    CALL SetStandardizeCasing WITH True
    CALL SetUpdateDomain WITH True
```

Step 5 — 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.

```
CALL VerifyEmail WITH EmailAddress RETURNING Success
```

Step 6 — Processing: Checking the Result Codes

If the VerifyEmail function returns a false value, an error has occurred. Check the GetResults function to determine the cause.

Result codes have replaced the old status and error codes with a comma-delimited string of four-letter codes. These codes will show the status of the last call to the VerifyEmail function, as well as the cause for any errors.

```
CALL GetResults RETURNING Results
Process Results
```

Step 7 — Retrieve Email Address Data

If the VerifyEmail function was called successfully, call the following functions to retrieve the returned data about the submitted email address.

```
IF Success Is True THEN
    CALL GetMailBoxName RETURNING MailBox
    CALL GetTopLevelDomain RETURNING TLD
    CALL GetTopLevelDomainDescription RETURNING
        TLDDesc
    CALL GetEmailAddress RETURNING EmailAddress
    CALL GetDomainName RETURNING DomainName
```

Step 8 — Termination: Destroying the Instance

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

```
Destroy emailObj
```

ParseFielded is faster and more accurate, but requires that the data is consistently delimited.

Important Information

Does the Data Quality Suite expire?

Your License String is good for the length of your subscription. When you renew your subscription to the Data Quality Suite, you will be issued a new license string.

The data on the Data Quality Suite DVD is valid for the following periods of time:

- Address Object: 90 days
- Canadian Add-on: 30 days
- Phone Object: 15 months
- Name Object: 18 months
- E-mail Object: 6 months

The Data Quality Suite is updated bimonthly, so subscribers will receive an updated DVD before your current data expires, based on the terms of your subscription. If you are not a subscriber, you should order the latest updated version of the Data Quality Suite, or consider upgrading your single purchase to a subscription.

How to purchase

To place an order for the Data Quality Suite, just call 1-800-MELISSA (1-800-635-4772).

Troubleshooting

If you need help using the products in the Data Quality Suite, please call us toll free at 1-800-MELISSA (1-800-635-4772). Our technical support staff is available Monday through Friday from 6 a.m. to 5 p.m. Pacific Standard Time.

To read about current known issues, visit: http://www.MelissaData.com/tech/tech.htm.

For our online support, visit: http://forum.MelissaData.com

You can also send an e-mail to Tech@MelissaData.com.