Reference Guide

PhoneObject

32/64 BIT

Multiplatform

MELISSA DATA®

Phone Object

Reference Guide

Copyright

Trademarks

MELISSA DATA CORPORATION
22382 Avenida Empresa
Rancho Santa Margarita, CA 92688-2112

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

E-mail: info@MelissaData.com
Web site: www.MelissaData.com

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

Document Code: PHORFG
Revision Number: 111006.227
Last Update: October 6, 2011

Dear Developer,

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

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

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

I look forward to hearing from you.

Best Wishes,

Raymond F. Melissa
President

Table of Contents

Phone Object

Phone Object allows Web sites and custom applications to verify phone numbers down to 7 and 10 digits, update area codes, and append data about the phone number.

Use Phone Object to:

- Verify U.S. or Canadian phone numbers down to 7 or 10 digits.
- Update the area code if it changed in the last year.
- Append data on the telephone line, distinguishing between landline, wireless numbers, or Voice Over IP (VOIP).
- Append data on the telephone owner, distinguishing between residential, business, or home office numbers.
- Parse the phone number into its components.
- Correct wrong or missing area codes (ZIP Code required).
- Check the distance from the phone number to the ZIP Code.

Entering Your Phone Object License

The license string is a software key that unlocks the functionality of the component. Without this key, the object does not function. You set the license string using an environment variable called MD_LICENSE. If you are just trying out Phone Object and have a demo license, you can use the environment variable MD_LICENSE_DEMO for this purpose. This avoids conficts or confusion if you already have active subscriptions to other Melissa Data object products.

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

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

Windows

Windows users can set environment variables by doing the following:

1. Select **Start > Settings**, and then click **Control Panel**.
2. Double-click **System**, and then click the **Advanced** tab.
3. Click **Environment Variables**, and then select either *System Variables* or *Variables for the user X*.
4. Click **New**.
5. Enter “MD_LICENSE” in the *Variable Name* box.
6. Enter the license string in the *Variable Value* box, and then click **OK**.

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

Linux/Solaris/HP-UX/AIX

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

```
export MD_LICENSE=A1B2C3D4E5
```

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

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

Using Phone Object

Verifying a Phone Number

1 Create an instance of Phone Object.

```
Set phonPtr as New Instance of PhoneCheck
```

2 Initialize the data files.

```
CALL Initialize WITH DataPath RETURNING Result
IF Result <> 0 Then
    CALL GetInitializeErrorString RETURNING ErrorString
     PRINT "Error: " & ErrorString
ENDIF
```

3 Pass a string containing a phone number and, optionally, a string containing a ZIP Code to the **Lookup** function. If the call is successful, use some or all of the functions shown below to retrieve the results.

```
CALL Lookup WITH Phone, ZIP
CALL GetResults RETURNING ResultCodes
Process ResultCodes
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 GetCountyName RETURNING CountyName
CALL GetCountyFips RETURNING CountyFips
CALL GetCountryCode RETURNING CountryCode
CALL GetLatitude RETURNING Latitude
CALL GetLongitude RETURNING Longitude
CALL GetDistance RETURNING Distance
CALL GetTimeZone RETURNING TimeZone
CALL GetTimeZoneCode RETURNING TimeZoneCode
```

Updating an Area Code after a Split

To update an area code after an area code split, use the **CorrectAreaCode** function.

```
CALL CorrectAreaCode WITH Phone,Zip RETURNING Result
IF Result is TRUE THEN
    CALL GetNewAreaCode RETURNING NewAreaCode
    CALL GetAreaCode RETURNING AreaCode
ELSE
    CALL GetResults RETURNING ResultCode
    Print ResultCode
ENDIF
```

Phone Object Functions

Initialize Phone Object

These functions initialize Phone Object and connect it to its data files.

Look Up Phone Numbers

These two functions look up a phone number, checking either for geographic data or an area code that was updated due to an area code split.

Retrieve Status Information

These function retrieve information about the most recent calls to the Lookup and CorrectAreaCode functions. The GetErrorCode and GetStatusCode functions have been deprecated in favor of the GetResults function.

Retrieve Phone Number Data

These functions retrieve information about the phone number passed via the last called to the Lookup function.

Retrieve GeoCode Data

The following functions retrieve geographic data about the phone number submitted to the Lookup function.

Computation Functions

The following functions do not require the data files to be initialized, but they can be used with the GetLatitude and GetLongitude functions to calculate distances and bearing between two records.

Initialize Phone Object

These functions initialize Phone Object and connect it to its data files.

SetLicenseString

The License String is a software key (supplied by the developer) that unlocks the full functionality of Phone Object.

Remarks

The license string is included with the documentation you received. If you have not purchased a license, call Melissa Data toll free at 1-800-MELISSA (1-800-635-4772) or send an email to sales@MelissaData.com.

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

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

For more information on setting the environment variable, see page 2 of this guide.

Input Parameters

The **SetLicenseString** function has one parameter:

LicenseString	A string value representing the software license key.

Return Value

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

Syntax

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

C

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

COM

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

Initialize

The **Initialize** function opens the needed data file and prepares the Phone Object for use.

Remarks

If the function returns any value other than 0, call the **GetInitializeErrorString** function to retrieve the cause of the failure.

Input Parameters

The **Initialize** function has one parameter:

DataPath — A String that contains the path to the location of the `mdPhone.dat`, `mdPhone.idx`, `mdAddr.dat` and `NPA.TXT` file.

Return Value

The **Initialize** function returns an integer value of 0 if successful.

Syntax

```
IntegerValue = object->Initialize(StringValue);
```

C

```
IntegerValue = mdPhoneInitialize(object, StringValue);
```

COM

```
IntegerValue = object.Initialize(StringValue)
```

GetInitializeErrorString

This function returns a descriptive string to describe the error from the Initialize function.

Remarks

The **GetInitializeErrorString** function returns a string describing the error caused when the Initialize function fails.

Syntax

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

C

```
StringValue = mdPhoneGetInitializeErrorString(object);
```

COM

```
StringValue = object.GetInitializeErrorString
```

GetBuildNumber

The **GetBuildNumber** function returns the current development release build number of the Phone Object.

Remarks

The word “DEMO” will be reported after the build number if no license string is provided, or if an incorrect license string is entered.

Input Parameters

None.

Return Value

The **GetBuildNumber** function returns the current development release build number of Phone Object.

Syntax

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

C

```
StringValue = mdPhoneGetBuildNumber(object);
```

COM

```
StringValue = object.GetBuildNumber
```

GetDatabaseDate

The **GetDatabaseDate** function returns a date value that represents the date of the phone data files.

Remarks

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

Input Parameters

None.

Return Value

The **GetDatabaseDate** function returns a value that represents the date of the phone data files. The COM object returns a date value, while the standard object returns a string value.

Syntax

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

C

```
StringValue = mdPhoneGetDatabaseDate(object);
```

COM

```
DateTime = object.GetDatabaseDate
```

GetLicenseExpirationDate

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

Remarks

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

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

Syntax

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

C

```
StringValue = mdPhoneGetLicenseExpirationDate(object);
```

COM

```
DateTime = object.GetLicenseExpirationDate
```

Look Up Phone Numbers

These two functions look up a phone number, checking either for geographic data or an area code that was updated due to an area code split.

Lookup

The **Lookup** function verifies the submitted phone number down to the first 7 or 10 digits and returns geographic information for the submitted phone number.

Remarks

On a FALSE return, the state and country code fields might still be populated if the area code is valid. Use the **GetResults** function to determine if the problem is a prefix error. If this is the case, you can use the GetState and **GetCountryCode** functions.

The return values of following functions are set by the **Lookup** function:

GetResults	GetAreaCode	GetPrefix
GetSuffix	GetExtension	GetCity
GetState	GetCountyName	GetCountyFips
GetCountryCode	GetDistance	GetLatitude
GetLongitude	GetMsa	GetPmsa
GetTimeZone	GetTimeZoneCode	

Input Parameters

Phone — A 24-character (maximum) string containing the phone number.

- Phone numbers from outside the U.S. and Canada will be not be validated.
- Toll free numbers will be parsed, but not verified.
- Phone Object will recognize a leading country code from U.S. and Canadian phone numbers (i.e. a “1”) and strip that digit before parsing the phone number.
- Any extensions (any consecutive digits after first ten) will be parsed out and returned by the **GetExtension** function.

ZIPCode — A five-character (optional) ZIP Code. If the ZIP Code is provided, the **GetDistance** function will contain the distance between the area code/prefix and the ZIP Code.

- If your programming language does not support optional parameters, you should pass an empty string (“”) to the function.

- The ZIPCode parameter only accepts U.S. ZIP codes and not Canadian Postal Codes.
- Because of phone number portability, the number returned by the **GetDistance** function may not be accurate.

Return Value

The **Lookup** function returns a value of (1) TRUE if the area code/prefix combination is valid and a value of (0) FALSE if it is invalid.

After calling the **Lookup** function, check the return values of the **GetResults** function. This will indicate the level of match, the type of phone number verified, and the cause of any errors.

Syntax

```
BooleanValue = object->Lookup(Phone[, ZIPCode]);
```

C

```
IntegerValue = mdPhoneLookup(object, Phone, ZIPCode);
```

COM

```
BooleanValue = object.Lookup(Phone[, ZIPCode])
```


CorrectAreaCode

The **CorrectAreaCode** function will return the correct area code for a phone number and ZIP Code combination to the NewAreaCode. The **CorrectAreaCode** function will also populate the values returned by **GetAreaCode** and **GetResults** functions.

Remarks

On a FALSE return, the **CorrectAreaCode** function will set the **GetResults** function to indicate the cause of failure.

Input Parameters

The **CorrectAreaCode** function has these parameters:

- PhoneNum – A 24 character (max) string containing the phone number.
- ZIPCode – A five-character VARIANT containing the ZIP Code.

Return Value

The **CorrectAreaCode** function returns a Boolean value of TRUE if the function call was successful, FALSE if there was an error.

Syntax

```
BooleanValue = object->CorrectAreaCode(PhoneNum, ZIPCode);
```

C

```
IntegerValue = mdPhoneCorrectAreaCode(object, PhoneNum,
ZIPCode);
```

COM

```
BooleanValue = object.CorrectAreaCode(PhoneNum, ZIPCode)
```

Retrieve Status Information

These function retrieve information about the most recent calls to the **Lookup** and **CorrectAreaCode** functions. The GetErrorCode and GetStatusCode functions have been deprecated in favor of the **GetResults** function.

GetResults

This function returns a comma-delimited string of four-character codes which detail the level of matching found for the current phone number and any errors that occurred during the last call to the **Lookup** function.

Remarks

The **GetResults** function is intended to replace the **GetStatusCode** and **GetErrorCode** functions, providing a single source of information about the last **Lookup** call and eliminating the need to call multiple functions to determine if a particular phone number was verified.

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

Code	Message	Details
PS01	10-digit match	The phone number was verified to the full 10-digit level.
PS02	7-digit match	Phone Object was able to verify the area code, the exchange and the first digit of the suffix.
PS03	Corrected Area Code	Corrected area code that was changed according to the ZIP Code it falls into.
PS04	Outside Demo Range	Phone number is outside the demo range.
PS05	Expired Database	The database has expired. Contact Melissa Data.
PS06	Updated Area Code	The area code has split and the new area code is returned by the **GetNewAreaCode** function. This area code was changed using area code/prefix data. Applies to new splits only.
PS07	Exchange Type - Cell Phone	The phone number belongs to a wireless phone.
PS08	Exchange Type - Not Special	The phone number belongs to a standard land line.
PS09	Exchange Type - VOIP	The phone number is assigned to a Voice Over IP (VOIP).
PS10	Phone Type: Residential	The phone number belongs to a residence.
PS11	Phone Type: Business	The phone number is assigned to a business.

Code	Message	Details
PS12	Phone Type: SOHO	The phone number is assigned to a small business or home office.
PE01	Bad Area Code	The area code does not exist in our database or contains non-numbers.
PE02	Blank Phone Number	Phone number was not populated.
PE03	Bad Phone Number	Too many or too few digits.
PE04	Multiple Match	Multiple Match (could not choose between 2 or more area codes as a bad or missing area code was encountered and the distance between the area codes was too close to choose one over the other).
PE05	Bad Prefix	The prefix does not exist in the database.
PE06	Bad ZIP Code	An invalid ZIP Code was entered.

Syntax

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

C

```
StringValue = mdPhoneGetResults(object);
```

COM

```
StringValue = object.Results
```

GetStatusCode (Deprecated)

This function returns the status code after a call to the **CorrectAreaCode** or **Lookup** function.

This function has been deprecated. You should use the **GetResults** function instead. See page 15 for documentation on this function.

Remarks

The **GetStatusCode** function returns a one-character string value set by a call to the **CorrectAreaCode** or **Lookup** function.

Possible return values from the **GetStatusCode** function are as follows:

Code	Definition
C	Corrected area code that was changed according to the ZIP Code it falls into.
D	Phone number is outside the demo range.
E	Expired database.
U	Updated area code (The area code has split and the new area code is returned by the **GetNewAreaCode** function. This area code was changed using area code/prefix data). Applies to new splits only.
X	Bad phone number.

Syntax

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

C

```
StringValue = mdPhoneGetStatusCode(object);
```

COM

```
StringValue = object.StatusCode
```


GetErrorCode (Deprecated)

This function returns the error code after an unsuccessful call to the CorrectAreaCode or the **Lookup** function.

This function has been deprecated. You should use the **GetResults** function instead. See page 15 for documentation on this function.

Remarks

The **GetErrorCode** function returns a 1-character string value containing the error code of the phone number string passed into the **CorrectAreaCode** or **Lookup** function.

Possible return values from the **GetErrorCode** function are as follows:

Code	Definition
A	Bad area code (area code does not exist in the database or the area code contains characters)
B	Blank (phone number is blank)
E	Bad Phone Number (too many or too few digits)

Code	Definition
M	Multiple Match (could not choose between 2 or more area codes as a bad or missing area code was encountered and the distance between the area codes was too close to choose one over the other)
P	Bad Prefix (this prefix does not exist in the database)
Z	Bad ZIP Code (an invalid ZIP Code was entered, for example, 00001)

If the **CorrectAreaCode** or **Lookup** function has not been called, or resulted in no error, the return value of this function will be Null.

Syntax

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

C

```
StringValue = mdPhoneGetErrorCode(object);
```

COM

```
StringValue = object.ErrorCode
```

Retrieve Phone Number Data

These functions retrieve information about the phone number passed via the last called to the **Lookup** function.

GetAreaCode

This function returns the area code of the phone number string that was passed to the **CorrectAreaCode** or **Lookup** function.

Remarks

The **GetAreaCode** function returns a three-character string value set by a call to the **CorrectAreaCode** or **Lookup** function.

If there are not enough digits for the area code, this function will be empty. If the area code/prefix combination has been split, the new area code will be in the **GetNewAreaCode** function. If a ZIP Code was also passed in and corrections were made to the area code, these corrections will appear in the **GetNewAreaCode** function. If the **CorrectAreaCode** or **Lookup** function has not been called, or the call resulted in an error, this function will be blank.

Syntax

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

C

```
StringValue = mdPhoneGetAreaCode(object);
```

COM

```
StringValue = object.AreaCode
```

GetNewAreaCode

This function returns the new area code if the phone number passed to the **CorrectAreaCode** function has undergone an area code split.

Remarks

The **GetNewAreaCode** function returns a three-character string value after a call to the **CorrectAreaCode** function.

Example: “714”

The **GetNewAreaCode** function will return a corrected or updated area code based on the phone number string passed into the **CorrectAreaCode** function.

An area code is corrected when it appears to be invalid for the ZIP Code passed in. Phone Object will change the area code based on the distance between valid area code/prefix combinations and the ZIP Code.

An updated area code is a new area code that is based on the input of an area code/prefix combination that has been split. If no new area code was found, then the **GetNewAreaCode** function will return the current area code.

If the **CorrectAreaCode** function has not been called, or resulted in an error, this function will return a null value.

Syntax

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

C

```
StringValue = mdPhoneGetNewAreaCode(object);
```

COM

```
StringValue = object.NewAreaCode
```

GetPrefix

This function returns the prefix (first three digits after the area code) of a phone number passed to the **CorrectAreaCode** function or the **Lookup** function.

Remarks

The **GetPrefix** function returns a three-character string value after a call to the **CorrectAreaCode** function or the **Lookup** function.

If the **CorrectAreaCode** function or **Lookup** function has not been called, or resulted in an error, this function will be blank.

Syntax

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

C

```
StringValue = mdPhoneGetPrefix(object);
```

COM

```
StringValue = object.Prefix
```

GetSuffix

This function returns the suffix (last four digits) of the phone number passed to the **Lookup** function.

Remarks

The **GetSuffix** function returns a four-character string value after a call to the **Lookup** function.

If the **Lookup** function has not been called, or resulted in an error, the return value of the GetSuffix function will be a blank string.

Syntax

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

C

```
StringValue = mdPhoneGetSuffix(object);
```

COM

```
StringValue = object.Suffix
```

GetExtension

This function returns the extension (if any) of the phone number passed to the **CorrectAreaCode** function or the **Lookup** function.

Remarks

The **GetExtension** function returns a maximum 10-character string value set after a call to the **CorrectAreaCode** function or the **Lookup** function.

If the **CorrectAreaCode** function or the **Lookup** function has not been called, or resulted in an error, this function will be blank.

Syntax

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

C

```
StringValue = mdPhoneGetExtension(object);
```

COM

```
StringValue = object.Extension
```

Retrieve GeoCode Data

The following functions retrieve geographic data about the phone number submitted to the **Lookup** function.

GetCity

This function returns the city name associated with the phone number passed to the **Lookup** function.

Remarks

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Syntax

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

C

```
StringValue = mdPhoneGetCity(object);
```

COM

```
StringValue = object.City
```

GetState

This function returns the state abbreviation associated with the area code passed to the **Lookup** function.

Remarks

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Syntax

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

C

```
StringValue = mdPhoneGetState(object);
```

COM

```
StringValue = object.State
```

GetCountyName

This function returns the name of the county associated with the phone number passed to the **Lookup** function.

Remarks

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Syntax

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

C

```
StringValue = mdPhoneGetCountyName(object);
```

COM

```
StringValue = object.CountyName
```

GetCountyFips

This function returns the five-digit county FIPS code associated with the phone number passed to the **Lookup** function.

Remarks

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

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

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

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

Syntax

C++

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

C

```
StringValue = mdPhoneGetCountyFips(object);
```

COM

```
StringValue = object.CountyFips
```

GetCountryCode

This function returns the two-character abbreviation that indicates the country of origin for the phone number passed to the **Lookup** function.

Remarks

Phone Object includes data for American and Canadian area codes. The two-character abbreviation “US” is returned for area codes in the United States, and the two-character abbreviation “CA” is returned for Canadian area codes.

Syntax

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

C

```
StringValue = mdPhoneGetCountryCode(object);
```

COM

```
StringValue = object.CountryCode
```

GetDistance

This function returns the distance in miles from the area code/prefix wire center to the ZIP Code centroid, if a ZIP Code has been passed to the **Lookup** function in addition to a phone number.

Remarks

The **GetDistance** function is a five-character (maximum) string value set by a call to the **Lookup** function.

The **GetDistance** function returns the distance between the centroids of the ZIP Code and wire center of the area code/prefix number passed into the **Lookup** function.

Because of phone number portability, the number returned by the **GetDistance** function may not be accurate.

If the **Lookup** function has not been called, resulted in an error, or no ZIP Code was passed in, this function will return “9999.”

Syntax

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

C

```
StringValue = mdPhoneGetDistance(object);
```

COM

```
StringValue = object.Distance
```

GetLatitude

This function returns the latitude of the NPA/NXX wire center for the phone number passed to the **Lookup** function.

Remarks

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Latitude is the geographic coordinate of the NPA/NXX wire center measured in degrees north or south of the equator. (NPA/NXX = area code/prefix).

The **GetLatitude** function returns a seven-character (max) string value after a call to the **Lookup** function. It is accurate to four decimal places. Phone numbers within the United States will always return a positive number.

If the **Lookup** function has not been called, or resulted in an error, this function will return 0.0.

Syntax

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

C

```
StringValue = mdPhoneGetLatitude(object);
```

COM

```
StringValue = object.Latitude
```

GetLongitude

This function returns the longitude of the NPA/NXX wire center for the phone number passed to the **Lookup** function.

Remarks

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Longitude is the geographic coordinate of the NPA/NXX wire center measured in degrees east or west of the Greenwich Meridian (NPA/NXX = area code/prefix).

The **GetLongitude** function returns a nine-character (max) string value after a call to the **Lookup** function. It is accurate to four decimal places and the negative sign is used to indicate a longitude in the United States.

If the **Lookup** function has not been called, or resulted in an error, this function will return 0.0.

Syntax

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

C

```
StringValue = mdPhoneGetLongitude(object);
```

COM

```
StringValue = object.Longitude
```

GetMsa

This function returns the Metropolitan Statistical Area (MSA) number associated with the phone number passed to the **Lookup** function.

Remarks

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

The **GetMsa** function returns a four-digit value after a call to the **Lookup** function.

If the **Lookup** function has not been called, or resulted in an error, this function's return value will be blank.

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Syntax

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

C

```
StringValue = mdPhoneGetMSA(object);
```

COM

```
StringValue = object.MSA
```

GetPmsa

This function returns the Primary Metropolitan Statistical Area (PMSA) number associated with the phone number passed to the **Lookup** function.

Remarks

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

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

Example: “5495” or “0”

If the **Lookup** function has not been called, or resulted in an error, this function value will be blank.

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Syntax

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

C

```
StringValue = mdPhoneGetPMSA(object);
```

COM

```
StringValue = object.PMSA
```

GetTimeZone

This function returns a string describing the time zone of the phone number passed to the **Lookup** function.

Remarks

The **GetTimeZone** function returns a string value describing the time zone where the phone number is located. The function will return one of the following strings:

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

If the input phone number cannot be corrected by the **Lookup** function, this string will be blank.

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Syntax

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

C

```
StringValue = mdPhoneGetTimeZone(object);
```

COM

```
StringValue = object.TimeZone
```

GetTimeZoneCode

This function returns a 1 or 2-digit code representing the time zone associated with the phone number passed to the **Lookup** function.

Remarks

Following are the possible values for the **TimeZoneCode** function:

Code	Zone	Code	Zone
4	Atlantic Time	10	Hawaii Time
5	Eastern Time	11	Samoa Time
6	Central Time	13	Marshall Island Time
7	Mountain Time	14	Guam Time
8	Pacific Time	15	Palau Time
9	Alaska Time		

If the input phone number cannot be corrected with the **Lookup** function, the return value will be blank.

Because of phone number portability, geographical information may not reflect the true location of the owner of the phone number for wireless and VOIP numbers.

Syntax

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

C

```
StringValue = mdPhoneGetTimeZoneCode(object);
```

COM

```
StringValue = object.TimeZoneCode
```

Computation Functions

The following functions do not require the data files to be initialized, but they can be used with the **GetLatitude** and **GetLongitude** functions to calculate distances and bearing between two records.

ComputeBearing

The **ComputeBearing** function returns a Bearing in degrees (-0 to 360) representing the compass direction from point 1 to point 2.

Remarks

You do not have to call the **Initialize** function before calling the **ComputeBearing** function.

Input Parameters

This function accepts four double-precision floating point numbers.

- **lat1** – latitude for point 1 [In Degrees (90 to –90)]
- **long1** – longitude for point 1 [In Degrees (180 to –180)]
- **lat2** – latitude for point 2 [In Degrees (90 to –90)]
- **long2** – longitude for point 2 [In Degrees (180 to –180)]

Return Value

The **ComputeBearing** function returns a double-precision floating point bearing based on input latitudes and longitudes.

Syntax

```
DoubleFloat = object->ComputeBearing(lat1, long1, lat2, long2);
```

C

```
DoubleFloat = mdPhoneComputeBearing(object, lat1, long1, lat2,
long2);
```

COM

```
DoubleFloat = object.ComputeBearing(lat1, long1, lat2, long2)
```

ComputeDistance

The **ComputeDistance** function returns a straight-line distance in miles between point 1 and point 2.

Remarks

You do not have to call the **Initialize** function before calling the **ComputeDistance** function.

Input Parameters

The function accepts four double-precision floating point numbers.

- **latitude 1** – latitude for point 1 [In Degrees (90 to –90)]
- **longitude 1** – longitude for point 1 [In Degrees (180 to –180)]
- **latitude 2** – latitude for point 2 [In Degrees (90 to –90)]
- **longitude 2** – longitude for point 2 [In Degrees (180 to –180)]

Return Value

The **ComputeDistance** function returns a double-precision floating point value representing the distance between two points based on the input latitudes and longitudes.

Syntax

```
DoubleFloat = object->ComputeDistance(lat1, long1, lat2,
long2);
```

C

```
DoubleFloat = mdPhoneComputeDistance(object, lat1, long1, lat2,
long2);
```

COM

```
DoubleFloat = object.ComputeDistance(lat1, long1, lat2, long2)
```