Reference Guide

GeoCoder Object

with GeoPoints

32/64 BIT

Multiplatform

MELISSA DATA®

GeoCoder 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@Melissa.com
Web site: www.Melissa.com

For the latest version of this Reference Guide, visit
http://wiki.Melissa.com/index.php?title=GeoCoder_Object.

Document Code: GCORFG
Revision Number: 200402.242
Last Update: April 2, 2020

Dear Programmer,

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

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

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

I look forward to hearing from you.

Best Wishes,

Raymond F. Melissa
President

Table of Contents

Chapter 1
Introduction to GeoCoder Object

GeoCoder Object retrieves the following information for a submitted 5-digit ZIP Code™ or 9-digit ZIP + 4®:

- County name
- County FIPS code
- Census Block
- Census Tract
- Core Based Statistical Area (CBSA) information
- Time zone
- Latitude and longitude of the ZIP centroid

With a 6-digit Canadian Postal Code, GeoCoder Object will return:

- Time zone
- Latitude and longitude of the Postal Code centroid

For example, if you submitted the ZIP + 4 of "92688-2112," the GeoCoder Object would return:

Function	Value
GetCensusBlock	4021
GetCensusTract	03206
GetCountyFIPS	06059
GetCountyName	Orange
GetLatitude	33.638915
GetLongitude	-117.603858

GeoCoder Object is compatible with many of the most popular programming languages, including Access, Active Server Pages, C, C++, Delphi, FoxPro, Power Builder, and Visual Basic. For sample source code that you can utilize in your own applications, visit our website at **www.Melissa.com**.

Geodetic System

The GeoCoder Object uses WGS 84 standard, an Earth-centered, Earth-fixed terrestrial reference system and geodetic datum.

GeoPoint Coding

Using multisource data, Melissa Data GeoPoint allows you to return latitude and longitude down to the rooftop of over 95% of all physical address in the United States. Within GeoPoint, there are two levels of accuracy: Rooftop and Interpolated Rooftop.

- **Rooftop** — These are the most accurate latitude and longitude available with coverage of approximately 105 million individual addresses.
- **Interpolated Rooftop** — These coordinates were computed using mathematical algorithms and street shape maps. While these coordinates are educated estimates, they should be quite accurate for the majority of the United States. There may be cases where the location of interpolated points is not very near the actual residence, mostly in rural areas with long streets and non-standard house numbering. However, these coordinates are still much more accurate than ZIP + 4 coordinates.

The basic version of GeoCoder Object lacks this GeoPoint feature. It is still accurate to the ZIP + 4 level.

1.1: Entering Your GeoCoder 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 GeoCoder 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 GeoCoder 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 longer true. GeoCoder 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.

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

1.2: Using GeoCoder Object

1 Create an instance of GeoCoder Object.

```
Create GeoCoder as New Instance of mdGeo
```

2 Set the path to the data files.

```
CALL SetPathToGeoCodeDataFiles WITH DataPath
```

3 Call the **InitializeDataFiles** function to connect the GeoCoder Object to its supporting data file.

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

4 First call the **SetInputParameter** method providing MAK, AddressKey, or Zip Code as a parameter. Then call **FindGeo()** to start a search. Finally call **GetOutputParameter("Results")** to see the result of the search.

```
IF MAK is provided
    CALL SetInputParameter WITH MAK, makValue
IF AddressKey is provided
    CALL SetInputParameter WITH AddressKey, AddressKeyValue
ELSE
    CALL SetInputParameter WITH Zip, ZipValue
    IF Plus4 is provided
        CALL SetInputParameter WITH Plus4, Plus4Value
CALL FindGeo
CALL GetOuputParameter(Results) RETURNING ResultCode
IF ResultCode CONTAINS "GS01", "GS02", "GS03", "GS05", OR
        "GS06" THEN
    CALL GetOutputParameter(CountyName) RETURNING CountyName
    CALL GetOutputParameter(CountyFIPSCode) RETURNING
CountyFIPSCode
    CALL GetOutputParameter(CensusBlock) RETURNING
CensusBlock
    CALL GetOutputParameter(CensusTract) RETURNING
CensusTract
    CALL GetOutputParameter(Latitude) RETURNING Latitude
    CALL GetOutputParameter(Longitude) RETURNING Longitude
    CALL GetOutputParameter(PlaceCode) RETURNING PlaceCode
    CALL GetOutputParameter(PlaceName) RETURNING PlaceName
```

```
        CALL GetOutputParameter(CBSACode) RETURNING CBSACode
        CALL GetOutputParameter(CBSALevel) RETURNING CBSALevel
        CALL GetOutputParameter(CBSATitle) RETURNING CBSATitle
        CALL GetOutputParameter(CBSADivisionCode) RETURNING
CBSADivisionCode
        CALL GetOutputParameter(CBSADivisionLevel) RETURNING
CBSADivisionLevel
        CALL GetOutputParameter(CBSADivisionTitle) RETURNING
CBSADivisionTitle
        CALL GetOutputParameter(TimeZone) RETURNING TimeZone
        CALL GetOutputParameter(TimeZoneCode) RETURNING
TimeZoneCode
    ELSE
        PRINT ResultCode
    ENDIF
```

Recent Changes to GeoCoder Object

July, 2019

Canadian Rooftop geocoding is now supported.

April, 2019

CensusKeyDecennial property was added to GetOutputParameter.

August, 2018

All data files have been combined into a single data file, “mdGeoCode.db3”

New Methods: FindGeo(), SetInputParameter(), and GetOutputParameter().

Chapter 2
GeoCoder Object Functions

2.1: GeoCoder Function List

2.1.1: Initialize GeoCoder Object

These methods initialize GeoCoder Object and connect it to its data files.

2.1.2: Geocoding ZIP Codes and Addresses

These methods retrieve the geographic data for multiple attributes (FindGeo method).

2.1.3: Retrieve Results

The following methods return information on the success of the last call to the ComputeDistance or FindGeo methods.

2.1.4: Retrieve Geocode Data

The following method returns the specified geo information from the last call to the FindGeo method.

2.1.5: Appendix A: Deprecated Methods

The following methods have been deprecated. While you can use this reference to continue using these deprecated methods, we strongly recommend you use the newer methods: SetInputParameter(), FindGeo(), and GetOutputParameter().

2.1.1: Initialize GeoCoder Object

These methods initialize GeoCoder Object and connect it to its data files.

GetBuildNumber

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

Remarks

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

Syntax

COM

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

C++

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

C

```
StringValue = mdGeoGetBuildNumber(object);
```

GetDatabaseDate

The **GetDatabaseDate** function returns a date value representing the date of the GeoCoder Object database. GeoCoder data expires nine months after the GeoCoder Object database date.

Remarks

If the **GetDatabaseDate** function is called before the **InitializeDataFiles** function is called and the data files are not in the installation directory, the object will return a value of “12/30/1899”.

Return Value

The **GetDatabaseDate** function returns a value representing the date of the GeoCoder Object database. The COM object returns a date value, while the standard object returns a string value.

Syntax

COM

```
DateTime = object.GetDatabaseDate()
```

C++

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

C

```
StringValue = mdGeoGetDatabaseDate(object);
```

GetExpirationDate

The **GetExpirationDate** function returns a date value representing the date your data files expire. This date allows you to confirm that the data files you are using are the latest ones.

Remarks

If the **GetExpirationDate** function is called before the **InitializeDataFiles** function is called, an exception will be reported.

Return Value

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

Syntax

COM

```
DateTime = object.GetExpirationDate()
```

C++

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

C

```
StringValue = mdGeoGetExpirationDate(object);
```

GetInitializeErrorString

This function returns a descriptive string to describe any error in a call to the **InitializeDataFiles** function.

Remarks

If called before the **InitializeDataFiles** function, this function will return an empty string.

Syntax

COM

```
StringValue = object.GetInitializeErrorString
```

C++

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

C

```
StringValue = mdGeoGetInitializeErrorString(object)
```

GetLicenseExpirationDate

Returns a string value containing the expiration date of the current license string.

Remarks

Call this function to determine when your current license will expire. After this date, GeoCoder Object will no longer function.

Syntax

COM

```
StringValue = object.GetLicenseExpirationDate
```

C++

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

C

```
StringValue = mdGeoGetLicenseExpirationDate(object)
```

InitializeDataFiles

The **InitializeDataFiles** function opens the required data files and prepares the GeoCoder Object for use.

Remarks

The **SetPathToGeoCodeDataFiles** function must be called prior to calling this function.

If this function returns any value other than No Error, call the **GetInitializeErrorString** function to determine the cause of the failure.

The GeoCoder Object data expires nine months after the date indicated by the **GetDatabaseDate** function. After this date, a “DataFile Expired” error is returned.

Return Value

The **InitializeDataFiles** function returns an integer value of 0 if successful, and a non-zero value if unsuccessful.

Return Value	Initialize Error String	Return Value	Initialize Error String
0	No error.	2	Insufficient memory to initialize.
-1	Not initialized.	3	County file error.
1	Invalid path to data.	4	Data File Expired.

Syntax

COM

```
IntegerValue = object.InitializeDataFiles()
```

C++

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

C

```
IntegerValue = mdGeoInitializeDataFiles(object);
```

SetLicenseString

The License String is a software key from Melissa that unlocks the full functionality of the component.

Remarks

The license string is used to unlock a component's full functionality. The license string is included with the documentation you received.

If you have not purchased a license, call Melissa toll free at 1-800-MELISSA (1-800-635-4772) or send an email to sales@Melissa.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 3 of this guide.

If the license string is not set, the component will operate in a demonstration mode (limited to Nevada ZIP Codes) and will return the string "DEMO" after the **GetBuildNumber** function.

Input Parameters

The **SetLicenseString** function has one parameter.

LicenseString A string value representing the license.

If the license string is already set as an environment variable, the input parameter is passed as an empty string.

Return Value

The **SetLicenseString** function returns a Boolean value of 0 (FALSE) or 1 (TRUE). It will return a FALSE Boolean value if the license string provided is expired or doesn't exist.

Syntax

COM

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

C++

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

C

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

SetPathToGeoCanadaDataFiles

This function passes a string value containing the path to the data files used by the **ComputeDistance** function to geocode Canadian postal codes.

Remarks

This function must be called prior to calling the **InitializeDataFiles** function. It is optional if you do not intend to geocode Canadian addresses. The value must contain a valid path to the directory that contains the Canadian geocoing data file: `mdGeoCanada.db`.

Syntax

COM

```
object.PathToGeoCanadaDataFiles = StringValue
```

C++

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

C

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

SetPathToGeoCodeDataFiles

This function passes a string value containing the path to the data files used by GeoCoder Object.

Remarks

This function must be called prior to calling the **InitializeDataFiles** function. The value must contain a valid path to the directory that contains the GeoCode data files, `mdGeoCode.db3`.

Syntax

COM

```
object.PathToGeoCodeDataFiles = StringValue
```

C++

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

C

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

WriteToLogFile

This function writes data to a log file.

Remarks

This function must be called prior to calling the **InitializeDataFiles** function. The value may contain a valid path to the directory you want to contain the log file. Otherwise the log file will be created in the current directory of the program. If the log file cannot be created or written to, this function will return false.

Log File Schema

```
CustomerID|YYYY-MM-DD hh:mm:ss|GS01 Count|GS02 Count|GS05
Count|GS06 Count|CheckSum
```

Syntax

COM

```
BooleanValue = object.WriteToLogFile(string LogFile)
```

C++

```
BooleanValue = object->WriteToLogFile(const *char LogFile);
```

C

```
IntegerValue = mdGeoWriteToLogFile(object, const *char LogFile);
```

2.1.2: Geocoding ZIP Codes and Addresses

These methods retrieve the geographic data for multiple attributes (**FindGeo** method).

ComputeBearing

The ComputeBearing Method returns a bearing of 0 to 360 degrees representing the compass direction from point 1 to point 2. To convert your bearing to an approximate compass direction, use the following table:

Direction	Degrees	Direction	Degrees
N	0 to 22.5, 337.5 to 0	**S**	157.5 to 202.5
NE	22.5 to 67.5	**SW**	202.5 to 247.5
E	67.5 to 112.5	**W**	247.5 to 292.5
SE	112.5 to 157.5	**NW**	292.5 to 337.5

Remarks

If an invalid latitude or longitude is entered, a value of 999 will be returned.

You do not have to call the **GetBuildNumber** function before calling the **FindGeo** function.

Input Parameters

The method 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 Method returns a double-precision bearing based on input latitudes & longitudes.

Syntax

COM

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

C++

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

C

```
Double Float = mdGeoComputeBearing (object, lat1, long1, lat2, long2);
```

ComputeDistance

The ComputeDistance Method returns a distance in miles between point 1 and point 2.

Remarks

If an invalid latitude or longitude is entered, a value of 999 will be returned.

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

Input Parameters

The method 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 ComputeDistance Method returns the distance in miles between two points based on the input latitudes & longitudes.

Syntax

COM

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

C++

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

C

```
Double Float = mdGeoComputeDistance (object, lat1, long1, lat2,
    long2);
```

FindGeo

This method performs a lookup on the attributes set by the **SetInputParameter** method. As such, it is required after the **SetInputParameter** method.

SetInputParameter is called to give **FindGeo** data to perform lookups for the **GetOutputParameter** method. These three methods work together and must be called in the same order at all times: **SetInputParameter** first, **FindGeo** second, and finally **GetOutputParameter**.

Syntax

COM

```
StringValue = object.FindGeo
```

C++

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

C

```
StringValue = mdGeoFindGeo(object)
```

SetInputParameter

This method sets the input data (in key-value pairs) for the **FindGeo** lookup method. As such, it is required before the **FindGeo** method.

SetInputParameter is called to give **FindGeo** data to perform lookups for the **GetOutputParameter** method. These three methods work together and must be called in the same order at all times: **SetInputParameter** first, **FindGeo** second, and finally **GetOutputParameter**.

Remarks

All attributes can be set at the same time. The object will do lookups in a top-down order, following the order of the key strings from most significant to least (First MAK, then AddressKey, followed by Zip, etc.) See the list below for the full ordering.

Input Parameters

The **SetInputParameter** function has the following parameters.

KeyString	A string value representing thekey.
ValueString	A string value representing the value.

The available key strings are below, ordered from most significant to least:

Significance	Key Strings	Key String Descriptions
1	**MAK**	A propietary unique key identifier for an address. This is derived from Address Checking.
2	**AddressKey**	An 11 digit key of the property. Based on the Zip5, Plus4 and two digit delivery code.
3	**Zip**	The 5-digit postal code.
4	**Plus4**	The 4-digit plus4 of the postal code.
5	**Latitude**	The latitude in degrees. This is the same function as **SetLatitude**.
6	**Longitude**	The longitude in degrees. This is the same function as **SetLongitude**.

Syntax

COM

```
BooleanValue = object.SetInputParameter(KeyString, ValueString)
```

C++

```
BooleanValue = object->SetInputParameter(KeyString,
    ValueString);
```

C

```
IntegerValue = mdGeoSetInputParameter(object, KeyString,
    ValueString);
```

2.1.3: Retrieve Results

The following methods return information on the success of the last call to the **ComputeDistance** or **FindGeo** methods.

GetResultCodeDescription

This function returns the description of the inputted Result Code. It can only be used through the Standard DLL.

It requires two values to be passed in, a Result Code and an enumerated option. If a string of Result Codes are inputted, only the first code will be used. The enumerated option will determine whether a short or long description will be returned.

Enumerated Value	Integer Value	Description
ResultCodeDescriptionLong	0	Returns a detailed description of the inputted result code.
ResultCodeDescriptionShort	1	Returns a brief description of the inputted result code.

Syntax

C++

```
StringValue = object->GetResultCodeDescription
(StringValue_ResultCode,ResultCdDescOpt);
```

C

```
StringValue = mdGeoGetResultCodeDescription
(object,StringValue_ResultCode,int);
```

GetResults

This function returns a string value containing status and error codes for the current record. Multiple codes are separated by commas.

Remarks

The **GetResults** function may return one or more four-character strings, separated by commas, depending on the result generated by the current record.

The possible values are:

Code	Short Desc.	Long Description
GS01	Geocoded to Street Level	The record was geocoded to the street (thoroughfare) level (Zip+4 for US, full postal code for CA).
GS02	Geocoded to Neighborhood Level	The record was geocoded to the neighborhood level (Zip+2 for US).
GS03	Geocoded to City Level	The record was geocoded to the city (locality) level (ZIP centroid for US, 3-digit postal code for CA).
GS05	Geocoded to Rooftop Level	The record was geocoded to the rooftop level.
GS06	Geocoded to Interpolated Rooftop Level*	The record was geocoded to the rooftop level using interpolation (educated estimations using street coordinates).
GE01	Invalid Postal Code	The submitted postal code is not in a valid format.
GE02	Postal Code Not Found	The submitted postal code was not found in the database.
GE03	Demo Mode	Geocoder is in Demo mode and a zip code outside the Demo range was detected.
GE04	Data Files Expired	Geocoder data files are expired. Please update with the latest data files.
GE05	License Not Enabled For Country	Geocoding for the country of input record is disabled for your License Key. Please contact your sales representative to enable.
GW01	Expiring License	Your License Key is expiring soon. Please contact your sales representative for a new License Key.
GW02	Usage Report Warning	Unable to report usage. The object will still continue to operate.

Code	Short Desc.	Long Description
GW03	Usage Report Error	Unable to report usage. Warning time exceeded. The object will not continue to operate.

* See *GeoPoint Coding* on page 2 for more information on interpolated roof-top coding.

If the location information in the current record was valid, this field will contain the value “GS01” at the very minimum and may include more of the “GS” codes. If the address could not be verified, the codes beginning with “GE” will indicate the reason or reasons why verification failed.

Syntax

COM

```
string = object.Results
```

C++

```
char = object->GetResults()
```

C

```
char = mdGeoGetResults(object)
```

2.1.4: Retrieve Geocode Data

The following method returns the specified geo information from the last call to the **FindGeo** method.

GetOutputParameter

This method returns the GeoCoded element requested and looked-up by the **FindGeo** method. As such, it is required after the **SetInputParameter** and **FindGeo** methods, in that order.

SetInputParameter is called to give FindGeo data to perform lookups for the **GetOutputParameter** method. These three methods work together and must be called in the same order at all times: **SetInputParameter** first, **FindGeo** second, and finally **GetOutputParameter**.

Remarks

The **FindGeo** method must be called prior to calling this method.

The possible element names you can request are as follows:

Element Name	Description
BlockSuffix	If the CensusKey is 15-characters, this will be empty. If the CensusKey is 16-characters, the last character is the BlockSuffix.
CBSACode	5-digit code for the CBSA.
CBSATitle	Title for the CBSA, if any.
CBSALevel	Level (metropolitan or micropolitan) of the CBSA, if any.
CBSADivisionCode	Code for the CBSA division, if any.
CBSADivisionTitle	Title for the CBSA division, if any.
CBSADivisionLevel	Division Level (metropolitan or micropolitan) of the CBSA division, if any.
CensusBlock	Census Block as defined by the U.S. Census Bureau.
CensusKey	15-digit string containing the concatenated County FIPS, Census Tract and Census Block.

Element Name	Description
CensusKeyDecennial	15-digit string containing the concatenated County FIPS, Census Tract, and Census Block for the most recent Decennial Census. The Decennial Census occurs every 10 years with the most recent available being 2010.
CensusTract	Census Tract as defined by the U.S. Census Bureau.
CountyFips	5-digit numeric Federal Information Processing Standard (FIPS) code.
CountyName	The name of the county.
CountySubdivisionCode	5-digit string representing the County Subdivision Code.
CountySubdivisionName	The County Subdivision Name.
ElementarySchoolDistrictCode	5-digit string representing the Elementary School District Code.
ElementarySchoolDistrictName	The Elementary School District Name.
Latitude	The latitude in degrees.
Longitude	The longitude in degrees.
PlaceCode	The area code located outside of a city but within the ZIP Code.
PlaceName	The area name located outside of a city but within the ZIP Code.
Results	Comma delimited status, error codes, and change codes.
SecondarySchoolDistrictCode	5-digit string representing the Secondary School District Code.
SecondarySchoolDistrictName	The Secondary School District Name.
StateDistrictLower	3-digit string representing the Lower State District Code.
StateDistrictUpper	3-digit string representing the Upper State District Code.
TimeZone	The Time Zone name.
TimeZoneCode	3-letter string representing the Time Zone Code.
UnifiedSchoolDistrictCode	5-digit string representing the Unified School District Code.

Element Name	Description
UnifiedSchoolDistrictName	The Secondary Unified District Name.

If the **FindGeo** method was not called first, the return value of this method will be an empty string.

Syntax

COM

```
StringValue = object.GetOutputParameter(ElementNameString)
```

C++

```
StringValue = object->GetOutputParameter(ElementNameString);
```

C

```
StringValue =
   mdGeoGetOutputParameter(object,ElementNameString);
```

2.1.5: Appendix A: Deprecated Methods

The following methods have been deprecated. While you can use this reference to continue using these deprecated methods, we strongly recommend you use the newer methods: **SetInputParameter()**, **FindGeo()**, and **GetOutputParameter()**.

GeoCode

This function obtains and sets the return values of the GeoCoder Object with Geographic and Census data, using an input ZIP Code and Plus4.

Remarks

If a 9-digit ZIP Code or 5-digit ZIP Code and Plus4 are used as parameters for the **GeoCode** function, the following will be done in order, if the previous fails:

1. Information will be obtained from the full ZIP and Plus4 combination. The **GetResults** function will return “GS01.”
2. If no information is found for the Plus4, the centroid for the ZIP+2 segment will be used (The ZIP Code & first two digits of the Plus4). The **GetResults** function will return “GS02.”
3. If the ZIP+2 segment cannot be found, the centroid for the 5-digit ZIP Code will be used. The **GetResults** function will return “GS03.”

Input Parameters

The **GeoCode** function has the following parameters:

Zip (Required)	The 5-digit ZIP Code, 9-digit ZIP + 4 code, or 6-digit Canadian Postal Code. A ZIP Code can consist of five digits to represent a 5-digit ZIP Code, or nine digits to represent a ZIP + 4 code. If the ZIP + 4ZIP + 4 is included here, a hyphen can be used to separate the two numbers, but is not required.
Plus4 (Optional)	The 4-digit Plus4 code if not included with the ZIP parameter.

When geocoding Canadian postal codes the full six-digit postal code must be included in the first parameter (Zip),

Return Value

The **GeoCode** function returns a Boolean value of 0 (FALSE) or 1 (TRUE).

A TRUE return will set the return values of the various functions described in the section beginning on page 23. On a FALSE return, check the **GetResults** function.

Syntax

COM

```
Boolean = object.GeoCode(Zip, Plus4)
```

C++

```
Boolean = object->GeoCode(Zip, Plus4);
```

C

```
Integer = mdGeoGeoCode(object, Zip, Plus4);
```

GeoPoint

Obtains and sets the return values of the GeoCoder Object with Geographic and Census data, using an input ZIP Code, Plus4 and delivery point code. If successful, the data will be accurate to the rooftop level, rather than just the ZIP + 4 code, as with the **ComputeDistance** function.

Remarks

If a five-digit ZIP Code, Plus4 and valid delivery point are used as parameters for the **GeoPoint** function, the following will be done in order, if the previous fails:

1. Rooftop-level information will be obtained for the full combination of ZIP Code, Plus4 and Delivery Point Code (Result Code “GS05”).
2. If the Delivery Point Code is not found, information will be returned for the centroid of the ZIP and Plus4 combination (Result Code “GS01”).
3. If no information is found for the Plus4, the centroid for the ZIP+2 (The ZIP Code & first two digits of the Plus4) segment will be used (Result Code “GS02”).
4. If the ZIP+2 segment cannot be found, the centroid for the 5-digit ZIP Code will be used (Result Code “GS03”)
5. Check the return value of the **GetResults** function to verify the level to which the record was coded.

Input Parameters

The **GeoPoint** function can use one of the two following sets of string values as parameters:

AddressKey	This is the preferred input. The 11-digit AddressKey contains all the information that the object requires. If the AddressKey input is used, do not use the other input parameters.

Or:

Zip	The 5-digit ZIP Code, 9-digit ZIP + 4 code, or 6-digit Canadian Postal Code. A ZIP Code can consist of five digits to represent a 5-digit ZIP Code, or nine digits to represent a ZIP + 4 code. If the ZIP + 4 is included here, a hyphen can be used to separate the two numbers, but is not required.
Plus4	The four-digit Plus4 code.
DeliveryPoint Code	The two-digit Delivery Point Code.

The AddressKey parameter is the preferred input. However either the AddressKey or the combination of Zip, Plus4, and DeliveryPoint Code are valid. For Canadian records, the Plus4 and DeliveryPoint Code parameters must be blank.

Refer to the syntax below for an example usage of both inputs.

Return Value

The **GeoPoint** function returns a Boolean value of 0 (FALSE) or 1 (TRUE).

A TRUE return will set the return values of the functions described in the section beginning on page 23. On a FALSE return, check the **GetResults** function.

Syntax

COM

```
Boolean = object.GeoPoint(AddressKey, "", "")

Boolean = object.GeoPoint(ZipCode, Plus4, DPC)
```

C++

```
Boolean = object->GeoPoint(AddressKey, "", "");

Boolean = object->GeoPoint(ZipCode, Plus4, DPC);
```

C

```
Integer = mdGeoGeoPoint(object, AddressKey, "", "");

Integer = mdGeoGeoPoint(object, ZipCode, Plus4, DPC);
```

GetCBSACode

This function returns the five-digit code for the Core Based Statistical Area (CBSA).

Remarks

Metropolitan and micropolitan statistical areas (metro and micro areas) are geographic entities defined by the U.S. Office of Management and Budget (OMB) for use by Federal statistical agencies in collecting, tabulating, and publishing Federal statistics. The term "Core Based Statistical Area" (CBSA) is a collective term for both metro and micro areas. A metro area contains a core urban area of 50,000 or more population, and a micro area contains an urban core of at least 10,000 (but less than 50,000) population. Each metro or micro area consists of one or more counties and includes the counties containing the core urban area, as well as any adjacent counties that have a high degree of social and economic integration (as measured by commuting to work) with the urban core.

The CBSA Code is a five-digit code for the specific CBSA associated with the location described by the submitted ZIP Code.

Syntax

COM

```
StringValue = object.CBSACode
```

C++

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

C

```
StringValue = mdGeoGetCBSACode(object);
```

GetCBSADivisionCode

This function returns the numeric code for the division within the Core Based Statistical Area containing the submitted ZIP Code.

Remarks

Some CBSA's are broken into parts known as divisions. In this case, the CBSA Division functions will also be populated. If not, these fields will be empty. Each division also has a Code, Level and Title.

Syntax

COM

```
StringValue = object.CBSADivisionCode
```

C++

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

C

```
StringValue = mdGeoGetCBSADivisionCode(object);
```

GetCBSADivisionLevel

This function returns the level description (metropolitan or micropolitan) for the division within the Core Based Statistical Area containing the submitted ZIP Code.

Remarks

Some CBSA's are broken into parts known as divisions. In this case, the return values of the CBSA Division functions will also be populated. If not, these values will be empty. Each division also has a Code, Level and Title.

Syntax

COM

```
StringValue = object.CBSADivisionLevel
```

C++

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

C

```
StringValue = mdGeoGetCBSADivisionLevel(object);
```

GetCBSADivisionTitle

This function returns the official U.S. Census Bureau's official name for the division within the Core Based Statistical Area containing the submitted ZIP Code.

Remarks

Some CBSA's are broken into parts known as divisions. In this case, the CBSA Division functions will also be populated. If not, these fields will be empty. Each division also has a Code, Level and Title.

Syntax

COM

```
string = object.CBSADivisionTitle
```

C++

```
char = object->GetCBSADivisionTitle();
```

C

```
char = mdGeoGetCBSADivisionTitle(object);
```

GetCBSALevel

This function returns the level description for the Core Based Statistical Area (CBSA), metropolitan or micropolitan.

Remarks

For more information on Core Based Statistical Areas, see the **GetOutputParameter** function on page 23.

Syntax

COM

```
StringValue = object.CBSALevel
```

C++

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

C

```
StringValue = mdGeoGetCBSALevel(object);
```

GetCBSATitle

Returns the official U.S. Census Bureau name for the Core Based Statistical Area (CBSA).

Remarks

For more information on Core Based Statistical Areas, see the **GetOutputParameter** function on page 23.

Syntax

COM

```
StringValue = object.CBSATitle
```

C++

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

C

```
StringValue = mdGeoGetCBSATitle(object);
```

GetCensusBlock

This function returns the Census Block number for the submitted ZIP + 4, returned after a successful call to the **ComputeDistance** or **GeoPoint** function.

Remarks

Census blocks, the smallest geographic area for which the Bureau of the Census collects and tabulates decennial census data, are formed by streets, roads, railroads, streams and other bodies of water, other visible physical and cultural features, and the legal boundaries shown on Census Bureau maps.

A Census Block Group is a cluster of blocks having the same first digit of their 3-digit identifying numbers within a Census Tract or Block Numbering Area (BNA). For example, Census Block Group 3 within a Census Tractor BNA includes all blocks numbered between 301 and 397. In most cases, the numbering involves substantially fewer than 97 blocks. Census Block Groups never cross Census Tract or BNA boundaries, however, they may cross the boundaries of county subdivisions, places, American Indian and Alaskan Native areas, urbanized areas, voting districts, and congressional districts. Census Block Groups generally contain between 250 and 550 housing units, with the ideal size being 400 housing units.

Census Blocks are small areas bordered on all sides by visible features such as streets, roads, streams, and railroad tracks, and by invisible boundaries such as city, town, township, county limits, property lines, and short, imaginary extensions of streets and roads.

The **GetCensusBlock** function returns a 4-character string value after a call to the **ComputeDistance** or **GeoPoint** function. The first digit is the Block Group and the last three characters (if any) are the Block Number.

Example:

"103A" (Block Group is 1 & Block Number is 3A)

If neither the **ComputeDistance** nor the **GeoPoint** function have been called, or if the function call resulted in an error, this function value will be empty.

Syntax

COM

```
string = object.CensusBlock
```

C++

```
char = object->GetCensusBlock();
```

C

```
char = mdGeoGetCensusBlock(object);
```

GetCensusTract

This function returns the Census Tract number for the submitted location, returned after a successful call to the **ComputeDistance** or **GeoPoint** function.

Remarks

Census Tracts are small, relatively permanent statistical subdivisions of a county. Census Tracts are delineated for all metropolitan areas (MA's) and other densely populated counties by local census statistical areas committees following Census Bureau guidelines (more than 3,000 Census Tracts have been established in 221 counties outside MA's).

The **GetCensusTract** function returns a four or six-character string value after a successful call to either the **ComputeDistance** or the **GeoPoint** function.

The Census Tract is usually returned as a 4-digit number. However, in areas that experience substantial growth, a Census Tract may be split to keep the population level even. When this happens, a 6-digit number will be returned.

If neither the **ComputeDistance** nor the **GeoPoint** function has been called, or if the function call resulted in an error, this value will be an empty string.

Syntax

COM

```
string = object.CensusTract
```

C++

```
char = object->GetCensusTract();
```

C

```
char = mdGeoGetCensusTract(object);
```

GetCountyFips

This function returns the County FIPS code for the submitted ZIP Code, after a successful call to either the **ComputeDistance** or the **GeoPoint** function.

Remarks

The Federal Information Processing Standard (FIPS) is a 5-digit code defined by the U.S. Bureau of Census. The first two digits are a state code and the last three indicate the county within the state.

The **GetCountyFips** function returns a 5-character string value set by a call to the **ComputeDistance** or **GeoPoint** function. It is accurate to the 9-digit level.

Example:

"06037" is the County FIPS for Los Angeles, CA.

In the example above, "06" is the state code for California and "037" is the county code for Los Angeles.

If neither the **ComputeDistance** nor the **GeoPoint** function has been called, or if the function call resulted in an error, this value will be an empty string.

Syntax

COM

```
StringValue = object.CountyFips
```

C++

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

C

```
StringValue = mdGeoGetCountyFips(object);
```

GetCountyName

This function returns the County name for the submitted ZIP Code after a successful call to either the **ComputeDistance** or the **GeoPoint** function.

Remarks

The **GetCountyName** function returns a 25-character (maximum) string after a successful call to either the **ComputeDistance** or the **GeoPoint** function.

If neither the **ComputeDistance** nor the **GeoPoint** function has been called, or if the function call resulted in an error, this value will be an empty string.

Syntax

COM

```
StringValue = object.CountyName
```

C++

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

C

```
StringValue = mdGeoGetCountyName(object);
```

GetErrorCode

This function returns error code triggered by an unsuccessful call to the **ComputeDistance** or **GeoPoint** function.

Since this has been deprecated you should use the **GetResults** function instead. See page 21 for documentation on this function.

Remarks

This function returns s a 1-character string value after an unsuccessful call to the **ComputeDistance** or **GeoPoint** function.

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

Error Code	Reason for Error
N	Record Not Found — Unable to locate the ZIP Code.
Z	Bad ZIP Code (an invalid ZIP Code was entered, for example, “abcde”)

If neither the **ComputeDistance** nor the **GeoPoint** function have been called, or if the function call did not result in an error, the return value of this function will be an empty string.

Syntax

COM

```
StringValue = object.ErrorCode
```

C++

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

C

```
StringValue = mdGeoGetErrorCode(object);
```

GetLatitude

This function returns the latitude for the submitted location after a successful call to either the **ComputeDistance** or the **GeoPoint** function.

Remarks

Latitude is the geographic coordinate of a point measured in degrees north or south of the equator. The GeoCoder Object uses the WGS-84 standard for determining latitude.

The **GetLatitude** function returns a character string representing a numeric value, set by a call to either the **ComputeDistance** or the **GeoPoint** function.

Since all U.S. ZIP Code latitude coordinates are north of the equator, this value will always be positive.

If neither the **ComputeDistance** nor the **GeoPoint** function has been called, or if the function call resulted in an error, this function will return “0.0.”

Syntax

COM

```
StringValue = object.Latitude
```

C++

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

C

```
StringValue = mdGeoGetLatitude(object);
```

GetLongitude

This function returns the longitude of the submitted location after a successful call to either the **ComputeDistance** or the **GeoPoint** function.

Remarks

Longitude is the geographic coordinate of a point measured in degrees east or west of the Greenwich meridian. The GeoCoder Object uses the WGS-84 standard for determining longitude.

The **GetLongitude** function returns an 11-character string value set by a call to either the **ComputeDistance** or the **GeoPoint** function. Its negative number indicates a point west of the Greenwich meridian.

If neither the **ComputeDistance** nor the **GeoPoint** function has been called, or if the function call resulted in an error, this function will return “0.0.”

Syntax

COM

```
StringValue = object.Longitude
```

C++

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

C

```
StringValue = mdGetGetLongitude(object);
```

GetPlaceCode

This function returns the Census Bureau place code associated with the location passed to either the **ComputeDistance** or the **GeoPoint** function.

Remarks

The **GetPlaceCode** function returns a seven-digit string value containing the Census place code for the submitted location.

ZIP Code boundaries sometime overlap with city limits and unincorporated areas. The ZIP Code may place a location within one city even though it is physically located within a neighboring area. The place code matches the ZIP + 4 code with the Census bureau's official name for that physical location.

If neither the **ComputeDistance** nor the **GeoPoint** function has been called, the submitted data was only a five-digit ZIP Code or if the function call resulted in an error, this function value will return an empty string value.

Syntax

COM

```
StringValue = object.PlaceCode
```

C++

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

C

```
StringValue = mdGeoGetPlaceCode(object);
```

GetPlaceName

This function returns the Census Bureau place name associated with location passed to either the **ComputeDistance** or the **GeoPoint** function.

Remarks

The **GetPlaceName** function returns a 60-digit string value containing the Census place name for the submitted location.

ZIP Code boundaries sometime overlap with city limits and unincorporated areas. The ZIP Code may place a location within one city even though it is physically located within a neighboring area. This function returns the Census bureau's official name for the ZIP + 4 code.

For example, the 92688 ZIP Code is located mostly within the city of Rancho Santa Margarita. However, it also contains parts of the unincorporated area of Los Flores. For these ZIP + 4 codes, the **GetCity** function of Address Object would return "Rancho Santa Margarita," but this function will return "Los Flores."

If neither the **ComputeDistance** nor the **GeoPoint** function has been called, the submitted data was only a five-digit ZIP Code or if the function call resulted in an error, this function will return an empty string value.

Syntax

COM

```
StringValue = object.PlaceName
```

C++

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

C

```
StringValue = mdGeoGetPlaceName(object);
```

GetStatusCode

The Status Code indicates how precisely the **ComputeDistance** or **GeoPoint** information was matched after a successful call to either of those functions.

Since this has been deprecated you should use the **GetResults** function instead. See page 21 for documentation on this function.

Remarks

The **GetStatusCode** function returns a one-character string value after a call to the **ComputeDistance** or **GeoPoint** function.

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

Status Code	Explanation
B	Record was coded to rooftop level.
A	Record was coded to interpolated rooftop level.*
9	Record was coded to the ZIP + 4 centroid (U.S.) or or the full 6-digit Postal Code level (Canada).
7	Record was coded to the ZIP + 2 centroid.
5	Record was coded to the 5-digit ZIP Code centroid (U.S.) or or the first 3-digit Postal Code level (Canada).
X	Record was not coded.
D	This object is in “Demonstration” mode and it detected the input of a ZIP Code outside the allowable range.
E	Expired.

* See *GeoPoint Coding* on page 2 for more information on interpolated rooftop coding.

If neither the **ComputeDistance** function nor the **GeoPoint** function have not been called, this function will return an empty string.

Syntax

COM

```
StringValue = object.StatusCode
```

C++

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

C

```
StringValue = mdGeoGetStatusCode(object);
```

GetTimeZone

This function returns the name of the time zone for the submitted location.

Remarks

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

This function will return one of the values under the Name column in the following table.

Code	Name	Code	Name
0	Military (APO or FPO)	9	Alaska Time
4	Atlantic Time	10	Hawaii Time
5	Eastern Time	11	Samoa Time
6	Central Time	13	Marshall Islands Time
7	Mountain Time	14	Guam Time
8	Pacific Time	15	Palau Time

The values under the Code column are returned by the **GetTimeZoneCode** function.

Syntax

COM

```
StringValue = object.TimeZone
```

C++

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

C

```
StringValue = mdGeoGetTimeZone(object);
```

GetTimeZoneCode

This function returns a numeric code for the time zone for the submitted location.

Remarks

For a list of the possible values returned by this function, see the table on page 43.

Syntax

COM

```
StringValue = object.TimeZoneCode
```

C++

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

C

```
StringValue = mdGeoGetTimeZoneCode(object);
```

Initialize

Since this has been deprecated you should use the **InitializeDataFiles** function. It may still be called instead of the **InitializeDataFiles** function if the GeoPoint functionality is not being used, but calling the **InitializeDataFiles** function is now the preferred method and required for using the **GeoPoint** function.

This function opens the required data files and prepares the GeoCoder Object for use.

Remarks

if this function returns any value other than 0, call the **GetBuildNumber** function to determine the cause of the failure.

GeoCoder data expires nine months after the GeoCoder Object database date. After this date, a “DataFile Expired” error is returned.

Input Parameters

The **Initialize** function has the following optional parameters:

DataPath	A string containing the path to the mdGeo.dat and mdGeo.lic file locations. The path parameter is optional. If the path is not given, the Initialize function will look for this data file in the same directory as the GeoObj.dll file (for Windows) or the target directory (for Unix).
IndexPath	A string containing the path to the mdGeo.idx and mdGeo.cty file locations. The path parameter is optional. If the path is not given, the Initialize function will look for this data file in the same directory as the GeoObj.dll file (for Windows) or the target directory (for Unix).

Return Value

The **Initialize** function returns the same values as the **InitializeDataFiles** function, show in the table on page 12.

Syntax

COM

```
IntegerValue = object.Initialize(DataPath, IndexPath)
```

C++

```
IntegerValue = object->Initialize(DataPath, IndexPath);
```

C

```
IntegerValue = mdGeoInitialize(object, DataPath, IndexPath);
```

SetLatitude

Use of the **SetLatitude** or **SetLongitude** functions will prevent records previously coded with GeoCoder Object from being counted towards your overall usage. By setting these properties to the values previously returned from GeoCoder Object, any records whose current values match the values set in the properties will not be counted against your total usage.

Syntax

COM

```
object.SetLatitude(StringValue)
```

C++

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

C

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

SetLongitude

Use of the **SetLatitude** or **SetLongitude** functions will prevent records previously coded with GeoCoder Object from being counted towards your overall usage. By setting these properties to the values previously returned from GeoCoder Object, any records whose current values match the values set in the properties will not be counted against your total usage.

Syntax

COM

```
object.SetLongitude(StringValue)
```

C++

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

C

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

SetPathToGeoPointDataFiles

This function passes a string value containing the path to the data files used by the **GeoPoint** function of GeoCoder Object.

Remarks

This function must be called prior to calling the **InitializeDataFiles** function. It is optional if you do not intend to use the **GeoPoint** function.

The value must contain a valid path to the directory that contains the GeoPoint data files, `mdGeoCode.db3`.

Syntax

COM

```
object.PathToGeoPointDataFiles = StringValue
```

C++

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

C

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

License Agreement

1. NOTICE. MELISSA DATA CORPORATION is WILLING TO LICENSE THE ENCLOSED SOFTWARE TO YOU, ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS LICENSE AGREEMENT. PLEASE READ THIS LICENSE AGREEMENT CAREFULLY BEFORE OPENING THE SEALED DISK PACKAGE. BY OPENING THIS PACKAGE (OR IN THE CASE OF DOWNLOADED SOFTWARE, YOU REQUEST UNLOCKING CODE FROM THE PUBLISHER) YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THESE TERMS WE ARE UNWILLING TO LICENSE THE SOFTWARE TO YOU, AND YOU SHOULD NOT OPEN THE DISK PACKAGE. IN SUCH CASE, PROMPTLY RETURN THE UNOPENED DISK PACKAGE AND ALL OTHER MATERIAL IN THIS PACKAGE ALONG WITH PROOF OF PAYMENT, TO THE AUTHORIZED DEALER FROM WHOM YOU OBTAINED IT FOR A FULL REFUND OF THE PRICE YOU PAID.

2. Ownership and License. This is a license agreement and NOT an agreement for sale. We continue to own the copy of the Software (including, but not limited to, object code, dynamic link libraries, and sample programs, together with the accompanying documentation contained in this package and all other copies that you are authorized by this Agreement to make collectively known as “Software”). Your rights to use the Software are specified in this Agreement, and we retain all rights not expressly granted to you in this Agreement. This Software is protected by U.S. copyright laws and

international treaties. Nothing in this Agreement constitutes a waiver of our rights under U.S. Copyright law or any other federal or state law or international treaty.

3. Permitted Uses. You are granted the following rights to the Software:

(a) Right to Install and Use.

(1) Standalone Computer - Single Installation You may install and use the Software on the hard disk drive of any single compatible computer that you own. However, you may not under any circumstances have the Software installed onto the hard drives of two or more computers at the same time, (nor may you install the Software onto the hard disk drive of one computer and then use the original CD-ROM on another computer). If you wish to use the Software on more than one computer, you must either erase the Software from the first hard drive before you install it onto a second hard drive, or else license an additional copy of the Software for each additional computer on which you want to use it.

(2) Network Use: If the single computer on which you install the Software is a network or Internet server, you may use the Software on any computer attached to the network, provided that it is only installed on the server. You may install and use this Software on a single file server regardless of the number of workstations attached to the network.

(b) Right to Copy. You may copy the Software for backup and archival purposes, provided that the original and each copy is kept in your possession, and that your installation and use of the Software does not exceed that allowed in part (a) above.

(1) Solely with the respect to the manual and Help files, you may make an unlimited number of copies (either in hard-copy or electronic form), provided that such copies shall be used only for internal purposes and are not republished or distributed beyond the licensee's premises.

(2) Copy, bundle, or redistribute the DEMO software with any commercial product (including books, CD-ROM, computer hardware, or software products). Your promotional and/or packaging materials must clearly disclose that the Software is copyrighted software of

Melissa Data, that no charge is made by Melissa Data or you for it, and that it is not a fully supported commercial version.

(c) Right to Modify. You may modify the Software and/or merge it into another computer program to the extent necessary for your own use on (a single computer or server as specified above); however, any portion of the Software merged into another computer program will continue to be subject to the terms of this Agreement. You may use and modify the source code version of those Software portions that the documentation identifies as sample code ("SAMPLE CODE"), provided you do not distribute the SAMPLE CODE or any modified version of the SAMPLE CODE, in source form.

(d) Right to Transfer. You may not rent, lend, or lease this Software. However, you may transfer this license to use the Software to another party on a permanent basis by transferring this copy of the License Agreement, at least one unaltered copy of the Software, and all documentation. You must, at the same time, either transfer to the other party or destroy all your other copies of the Software or destroy all of your copies. Such transfer of possession terminates your license from us. Such other party shall be licensed under the terms of this Agreement upon its acceptance of this Agreement by its initial use of the Software. If you transfer the Software, you must remove the Software from your hard disk and you may not retain any copies of the Software for your own use.

4. Prohibited Uses. You may not, without written permission from us:

(a) Use, copy, modify, merge, or transfer copies of the Software or documentation except as provided in this Agreement;

(b) Use any backup or archival copies of the Software (or allow someone else to use such copies) for any purpose other than to replace the original copy in the event it is destroyed or becomes defective;

(c) Disassemble, decompile or reverse engineer, or in any manner decode the Software for any reason;

(d) Distribute, Sublicense, lease, or rent the Software, Data files and/or Dynamic Link Libraries of the Software.

(e) Expose the interfaces of the Software through your application (e.g. an OCX, DLL, class library, etc.).

5. Limited Warranty. We make the following limited warranties, for a period of 180 days from the date you acquired the Software from us.

(a) Media. The disks and documentation in this package will be free from defects in materials and workmanship under normal use. If the disks or documentation fail

to conform to this warranty, you may, as your sole and exclusive remedy, obtain a replacement free of charge if you return the defective disk or documentation to us with a dated proof of purchase.

(b) Software. The Software in this package will materially conform to the documentation that accompanies it. If the Software fails to operate in accordance with this warranty, you may, as your sole and exclusive remedy, return all of the Software and the documentation to the authorized dealer from whom you acquired it, along with a dated proof of purchase, specifying the problem, and we will provide you with a new version of the Software or a full refund at our election.

(c) WARRANTY DISCLAIMER. WE DO NOT WARRANT THAT THIS SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT ITS OPERATION WILL BE UNINTERRUPTED OR ERROR-FREE. WE EXCLUDE AND EXPRESSLY DISCLAIM ALL EXPRESS AND IMPLIED WARRANTIES NOT STATED HEREIN, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

6. Termination. This license and your right to use this Software automatically terminate if you fail to comply with any provisions of this Agreement, destroy the copies of the Software in your possession, or voluntarily return the Software to us. Upon termination you will destroy all copies of the Software and documentation. Otherwise, the restrictions on your rights to use the Software will expire upon expiration of the copyright to the Software.

7. Miscellaneous Provisions. This Agreement will be governed by and construed in accordance with the substantive laws of California. This is the entire agreement between us relating to the contents of this package, and supersedes any prior purchase order, communications, advertising or representations concerning the contents of this package. No change or modification of this Agreement will be valid unless it is in writing, and is signed by us.