# Table of Contents

## Part 1: MapMarker User Guide

**Chapter 1: Introduction** ......................................................... 9  
- What Is Geocoding? ............................................................... 10  
- What Is MapMarker? .............................................................. 10  
- MapMarker Standard or MapMarker Plus .................................. 11  
- MapMarker Documentation Set ............................................... 12  
- MapMarker Streets ............................................................... 14

**Chapter 2: Installation and Upgrade Procedures** .......................... 15  
- System Requirements ............................................................ 16  
- System Recommendations ..................................................... 16  
- Installation Overview ........................................................... 17  
- Installation Procedure .......................................................... 18  
- Shared Installation .............................................................. 23  
- Unlocking DPV ........................................................................... 25  
- MDAC 2.7 and MapMarker ODBC Support for Remote Databases ......... 25  
- Installing MapMarker Server, OCX, and APIs .............................. 26  
- Starting MapMarker ................................................................. 27  
- Removing MapMarker From Your System .................................... 27  
- Installation Considerations ..................................................... 27  
- Creating Data Subsets ............................................................. 28  
- Moving Data and Program Files ............................................... 30

**Chapter 3: Geocoding with MapMarker** ....................................... 31  
- Before You Geocode .................................................................. 32  
- Running MapMarker ................................................................. 37  
- Identifying Address Columns .................................................. 37  
- Geocoding Your Table Automatically ........................................ 40  
- Geocoding Your Table Interactively ......................................... 41  
- Making the Most of Interactive Geocoding ................................ 43  
- Displaying Geocoded Records in MapInfo ................................... 44  
- Re-Geocoding a Table ............................................................... 45  
- Setting Geocoding Preferences ............................................... 45  
- Saving Settings to a Table ....................................................... 50

**Chapter 4: Match Settings and Strategies** .................................... 51  
- MapMarker's Geocoding Model .................................................. 52  
- Match Settings ......................................................................... 53  
- Choosing Match Strategies ..................................................... 53  
- Picking Your Strategy: An Example .......................................... 54  
- Impact of Relaxing Match Conditions ....................................... 56
# Table of Contents

MapMarker Geocoder Control Events and Methods ........................................... 125
Client/Server Geocoding and the Internet ...................................................... 127

Chapter 10: OLE Automation API ................................................................. 129
  Creating a Custom Geocoding Control ....................................................... 130
  Summary of Geocoder Control Properties, Events, and Methods ..................... 131
  OLE Automation Methods ........................................................................... 135
  OLE Automation Objects ........................................................................... 171
  Programming Usage Example ..................................................................... 174

Chapter 11: OLE Automation API for ColdFusion ............................................. 185
  OLE Automation Methods for ColdFusion .................................................. 186
  OLE Automation Objects ........................................................................... 200

Chapter 12: OLE Automation API – For ASP/VBScript .................................... 201
  ASP/VBScript Methods .............................................................................. 202
  ASP Sample Application ............................................................................. 224

Chapter 13: MapMarker Server RPC API ......................................................... 229
  MapMarker Server RPC Interface ............................................................... 230
  Using the RPC API to Build a Remote Geocoding Application ..................... 230
  RPC Server Error Codes ........................................................................... 230
  API Function Calls ................................................................................... 231

Chapter 14: MapMarker GeoEngine API ........................................................ 265
  Creating a Geocoding Application .............................................................. 266
  Notes for MapBasic and Visual Basic Programmers .................................... 277
  Sample Code ............................................................................................. 278
  MapMarker Geocoding Engine Function Calls .......................................... 278

Chapter 15: Data Dictionary API ..................................................................... 337
  Required and Optional Fields .................................................................... 338
  User Dictionary File Format ...................................................................... 339
  Data Dictionary Interface .......................................................................... 340

Part III: Appendixes ....................................................................................... 342

Appendix A: Data Structures ......................................................................... 343

Appendix B: MapMarker Preferences ............................................................ 369
  Setting System Preferences ....................................................................... 370
  Startup Preferences ................................................................................... 370
  Dictionary Preferences .............................................................................. 371
  Offset Preferences ..................................................................................... 372
  Datum Preferences ..................................................................................... 372
  Maps Preferences ....................................................................................... 373
<table>
<thead>
<tr>
<th>Appendix C: Understanding Datums</th>
<th>375</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of Datums</td>
<td>376</td>
</tr>
<tr>
<td>NAD83 and the MapMarker Address Dictionary</td>
<td>376</td>
</tr>
<tr>
<td>How MapMarker Determines Which Datum to Use</td>
<td>377</td>
</tr>
<tr>
<td>Datums and ODBC Tables</td>
<td>378</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix D: Creating a Map Catalog By Hand</th>
<th>379</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manually Creating a MapInfo Map Catalog</td>
<td>380</td>
</tr>
<tr>
<td>Making a Remote Table Mappable</td>
<td>381</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix E: Frequently Asked Questions</th>
<th>383</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix F: MapMarker Utility Programs</th>
<th>387</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindAddress</td>
<td>388</td>
</tr>
<tr>
<td>Append</td>
<td>389</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix G: MapMarker Program Files</th>
<th>393</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix H: MapMarker GeoEngine Error Codes</th>
<th>395</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix I: Developer Information</th>
<th>405</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapMarker Registry Entries</td>
<td>405</td>
</tr>
<tr>
<td>Required Files for Custom Applications</td>
<td>406</td>
</tr>
</tbody>
</table>

| Appendix J: U.S. ZIP Code Ranges | 411 |

| Appendix K: Street Suffix Abbreviations | 415 |

| Index | 421 |

© 2004 MapInfo Corporation. All rights reserved.
Part 1: MapMarker User Guide

User Guide Topics:

- Introduction .................................................. 9
- Installation and Upgrade Procedures ..................... 15
- Geocoding with MapMarker .............................. 31
- Match Settings and Strategies ............................ 51
- Result Codes .................................................. 61
- Specialized Geocoding Features ......................... 69
- Geocoding Remote Tables .............................. 93
- User Dictionaries ........................................... 105
Welcome to MapMarker, MapInfo’s premier address matching product. MapMarker enables you to assign geographic coordinates to large tables of U.S.-based address records in a single session. In addition to being a powerful geocoder, MapMarker is also a suite of tools that allows you to standardize your U.S. addresses, add spatial information and create points for your records, develop standalone or client/server custom geocoding applications, and embed MapMarker functionality in existing applications.

In this chapter:

- **What Is Geocoding?** ......................................................... 10
- **What Is MapMarker?** ...................................................... 10
- **MapMarker Standard or MapMarker Plus** ......................... 11
- **MapMarker Documentation Set** .......................................... 12
- **MapMarker Streets** ......................................................... 14
What Is Geocoding?

Geocoding is the process of assigning geographic coordinates to data that contain addresses. The coordinates assigned to each address turn each record into a geographic object that can be displayed on a map in either MapInfo Professional or via MapXtreme.

Visualizing your records on a map can make the relationships among your data clearer. You can display your geocoded records against a street map, a ZIP Code centroid map, a county map—whatever is most appropriate to your needs. You can then use the wide variety of functions available in MapInfo’s mapping software to perform querying, create thematic maps, create territories, and perform many other types of geographic analysis.

What Is MapMarker?

MapMarker is MapInfo Corporation’s powerful geocoding tool. It can geocode large tables of U.S.-based street addresses in a single pass. that is the first step toward mapping and analyzing your business data. MapMarker adds geographic coordinates to every record in your database that it matches against its comprehensive Address Dictionary, a database of USPS street addresses, street geometry and the latest ZIP+4 centroids.

MapMarker assigns coordinates to an address based on how well it matched in the Address Dictionary. The precision of the match can vary. For each address you geocode, you may get back a single perfect, street-level match, a list of street-level match candidates from which you choose the best match, or a less precise ZIP Code centroid match, where the point would be located near the center of the area of a ZIP Code. In the case of a ZIP+4 centroid match, the location of the point corresponds to the address which is closest to the mid-address of the ZIP+4 address range.

To identify the match precision, MapMarker also returns a result code for each address that it geocodes. The precision that you require for your geocoded records depends on how you plan to use your data.

MapMarker provides its geocoding capabilities in the MapMarker desktop application, the MapMarker server, and in developer tools that allow you to add geocoding functionality to your own desktop or Web application. The next sections briefly the different parts of the product.

MapMarker Desktop Application

The MapMarker desktop application can be installed on a single machine, or on a network to be shared by other users. The desktop application gives you a great deal of control over the geocoding process. For example, you can geocode a portion of your table using the Quick Geocode feature, or geocode a large database of addresses in batch mode.
Other features include the ability to:

- Geocode interactively (you make the final decision if it is a match) to maximize the number of matches and to control error rate.
- Geocode to street addresses ZIP Code centroids, or street intersections.
- Identify result codes by the type of match to tell at a glance the street address matches from the ZIP Code centroid matches.
- Standardize addresses to meet USPS CASS requirements using MapMarker Plus Address Dictionary.
- Use Delivery Point Validation (DPV) while geocoding in CASS mode to verify whether a geocoded address exists.
- Geocode remote tables via ODBC.

**Beyond the Desktop**

In addition to being a desktop application, MapMarker is also a server product and a developer tool.

Using the MapMarker server, multiple simultaneous geocoding requests can be served from a single MapMarker engine.

The MapMarker developer tools include an OCX for adding geocoding functionality to your own applications. And if the OCX doesn't provide exactly what you need, there is an OLE Automation API for creating one yourself. For those who work in C, the complete geocoding engine API is provided as well.

**MapMarker Standard or MapMarker Plus**

The MapMarker product is available in two versions: Standard or Plus. If you purchased MapMarker Standard, you receive, in addition to the desktop application, the MapMarker geocoding engine, the standard Address Dictionary, MapMarker Server and Geocoder Control (OCX), utility programs, and sample applications. MapMarker Standard is released twice a year.

If you purchased the MapMarker Plus product, your package contains everything in the MapMarker Standard product with the exception of the standard Address Dictionary. In the Plus product you receive the Plus Address Dictionary, an enhanced dictionary of addresses that geocodes even more records. The Plus Address Dictionary meets the U.S. Postal Service requirements for CASS certification for address standardization and bulk mail discounts. MapMarker Plus with the Plus Address Dictionary is released quarterly.
MapMarker Documentation Set

The documentation set for MapMarker Standard and MapMarker Plus provides both hard copy and online resources to help you make the most of this geocoding product. The set includes:

- MapMarker User Guide (includes a Programmer Reference) in print and in PDF format
- MapMarker Installation Guide in PDF format
- Online Help for both the MapMarker product and installer

MapMarker User Guide

This book is a three-part product guide designed to help you use MapMarker to the fullest and develop custom geocoding applications that call MapMarker. The following is a brief description of how the book is organized

Part I: MapMarker User Guide

Part I of the book introduces you to the product and documentation set, gives installation instructions for the product components, and explains how to use the MapMarker desktop application. This includes:

- geocoding in MapMarker
- configuring match settings for optimum geocoding results
- an explanation of result codes
- geocoding remote tables
- creating a customized user dictionary.

It also contains a chapter on more specialized features, including CASS geocoding and Delivery Point Validation (DPV); finding airports, highway exits, and single addresses; batch geocoding, and table attribution.

Part II: MapMarker Developer Guide

Part II of the book is a developer guide that describes the application programming interfaces (APIs) that come as part of MapMarker for developers to create client/server or standalone geocoding applications. It includes chapters on:

- client/server geocoding
- OLE Automation API
- OLE Automation API for ColdFusion
- OLE Automation API – For ASP/VBScript
- MapMarker Server RPC API
- MapMarker GeoEngine API
- Data Dictionary API

Part III: Appendixes

Part III of the User Guide focuses on reference information for both the MapMarker desktop application and the developer tools. It includes:

- Data Structures—Provides descriptions of the data structures used in the geocoding engine.
- MapMarker Preferences—Provides an explanation of the different preferences and how they are used.
- Understanding Datums—Gives an explanation of how datums are used in MapMarker.
- Creating a Map Catalog—Describes how to create a Map Catalog manually so that you can make remote tables mappable.
- MapMarker Utility Programs—Explains the FindAddress and Append utilities.
- MapMarker GeoEngine Error Codes—Provides a complete list and description of the error codes used in the GeoEngine.

In addition, the following appendixes are available in the PDF version of the MapMarker User Guide:

- MapMarker User Interface Reference—A reference of all the commands and dialog boxes in MapMarker.
- Frequently Asked Questions—A list of frequently asked questions.
- MapMarker Program Files—A list of the files that are installed with MapMarker.
- Developer Information—Defines the registry entries that are created when MapMarker is installed.
- Street Suffix Abbreviations—A list of USPS standard abbreviations for words that frequently appear in street addresses.

MapMarker Installation Guide

This online reference walks you through the product installation process from beginning to end.

MapMarker Release Notes

The MapMarker Release Notes contain information on data vintages and data file sizes, as well as late-breaking information on the product.

Publications on the Web

The MapMarker User Guide, Installation Guide, and Release Notes are available for download on the MapInfo Web site (www.mapinfo.com). To access them, from the MapInfo home page, click the Support and Training tab; click the Documentation link; and then click Publications.

Online Help

In addition to the User Guide and Installation Guide, MapMarker Standard and MapMarker Plus include Online Help for both the product and the installer. Online Help is instantly available while you are running MapMarker or the installation wizard. To access Help, either choose the Help menu, press the F1 key, or click the Help button for help about a dialog box.
**MapMarker Streets**

Once you have geocoded your table and are ready to display it in MapInfo Professional, you will likely want to add other layers of information to your map to give your records a geographic reference. MapMarker Streets is a U.S. network of fast displaying streets, highways, municipal boundaries, water features, and points of interest to complement your geocoded data. MapMarker Streets ships with MapMarker. MapMarker Plus Streets ships with MapMarker Plus.
This chapter contains instructions for installing MapMarker Plus 10.0 or upgrading to version 10.0 from a previous release. Refer to this chapter for step-by-step instructions and other issues related to installation.

In this chapter:

- System Requirements ................................................................. 16
- System Recommendations ........................................................ 16
- Installation Overview ................................................................. 17
- Installation Procedure ................................................................. 18
- Shared Installation ......................................................................... 23
- Unlocking DPV ............................................................................ 25
- MDAC 2.7 and MapMarker ODBC Support for Remote Databases .......... 25
- Installing MapMarker Server, OCX, and APIs .................................... 26
- Starting MapMarker ....................................................................... 27
- Removing MapMarker From Your System ......................................... 27
- Installation Considerations .......................................................... 27
- Creating Data Subsets .................................................................. 28
- Moving Data and Program Files ..................................................... 30
System Requirements

MapMarker is a 32-bit program for Windows 2000 with SP4, Windows XP with SP1, and NT 4.0 with SP6. You must have Internet Explorer 5.0 or later with SP2 installed or the MDAC installer does not run properly. The minimum system requirements are a 90 Mhz Pentium® processor or equivalent with a minimum of 32 MB RAM.

The MapMarker software requires about 25 MB of available hard drive space. The Plus Address Dictionary requires approximately 1.35 GB (for nationwide data) of disk storage space.

System Recommendations

Please note the following recommendations in order to maximize the performance of your MapMarker Plus installation.

Storing Data on Your Hard Drive

To maximize MapMarker’s performance, we recommend that the Address Dictionary data be stored on a hard drive. The data resides in the data directory under the MapMarker program directory (default is c:\Program Files\mapinfo\mapmarker\data).

If space is limited, you can create subsets of the most frequently accessed data and copy them to your hard drive. See Creating Data Subsets on page 28.

Using Data from CD-ROM

If you use the data directly from CD-ROM, we recommend you configure a disk cache to improve performance. Refer to your disk caching software documentation for instructions.

Performance Tuning

To get the most out of MapMarker’s geocoding capabilities, consider these tips for optimizing your system.

- Use the fastest processor available to you.
- Have enough memory so that the operating system can allocate some memory to your disk cache.
- Copy the Address Dictionary to your hard drive.
- Sort your table by ZIP Code.
- Choose exact match criteria for all (house number, street name, city name, ZIP Code).
- Do not create points automatically.
Installation Overview

The MapMarker Plus product comes on three CD-ROMs and includes the software and data. The installer installs all the product components in one easy procedure. The installer also handles all upgrade procedures and registry keys. The components installed during a complete installation are:

- MapMarker Plus software
- MapMarker Plus Address Dictionary
- MapMarker Client (only for shared installations)
- MapMarker Developer Tools

The custom installation enables you to choose whether to install the Data files or leave them on the CD, Client (asked when the user chooses “Shared”), and Developer Tools, and where you want to install them.

Address Dictionary

The MapMarker Plus Address Dictionary is installed during the installation of the MapMarker Plus software. You can install individual states of data, the entire United States, ZIP+4 centroids, or an evaluation copy of the product.

During the installation, you will be asked to supply a data unlocking code for the data that you purchased. To obtain the code, call the MapMarker License Authorization Desk between 8 a.m. and 7 p.m. ET, Monday through Friday. You do not need to call if you choose the free evaluation version.

When you enter your unlocking code, the installer places a license on your system that matches your purchase and entitles you to access the appropriate data. All of the individual state license files are now combined into a single license file with an .L30 extension. The license file(s) is located in the \data directory after installation.

You may elect to install only a portion of the data set during the installation. To do this, select the Custom install option during the installation procedure. When you reach the Custom Setup dialog, choose which states you want to remain on the CD.

Delivery Point Validation (DPV) Data

Customers who purchased the Delivery Point Validation (DPV) option must unlock the DPV data in a separate installation task using the DPV Unlocker application. This application is installed as part of MapMarker and is available in the MapMarker directory.

Upgrading to MapMarker Plus 10.0

Upgrading from MapMarker Plus 10.0 from MapMarker Plus 9.x does not require a new serial number or unlocking code as long as you purchased the same data with 10.0 as you did for 9.x. If you purchased different data, or if you purchased the new DPV data, then you will be required to enter a new unlocking code.
To upgrade, run the installer as you normally would for a new installation. At the Welcome dialog, a Question dialog will appear, asking you if you want to upgrade your existing version of MapMarker. Click Yes to upgrade. Your existing installation will be removed, and the installer will prepare to install version 10.0. If you purchased different data, or the new DPV data, the next dialog that displays is the Data Unlocking dialog. Enter your new unlocking code, and proceed with the installation.

If you did not purchase any new data, the installation continues at the Choose Installation Locations dialog. Go to step

If you elect not to upgrade, the installer requires that you install your new version of MapMarker in a location separate from your older version. You will not be able to overwrite an existing installation of MapMarker software or data.

Start the install program by running LAUNCH.EXE from CD 1 of 3. The MapMarker Installation Options screen directs you to install the product.

### Installation Procedure

To install MapMarker:

1. Place the MapMarker CD 1 of 3 in your CD-ROM drive.
   
   If the installation does not automatically start, from the Windows 2000/XP or NT 4.0 **START** button, choose **RUN**. From the Run dialog, type `D:\LAUNCH.EXE` in the Open command box, where D is the drive letter of your CD-ROM. Click **OK**. The MapMarker Installation Options screen displays on your screen.

2. Click the **INSTALL MAPMARKER...** button.
   
   The Welcome dialog displays.

3. At the Welcome dialog, click **NEXT** to proceed. After reading the License Agreement, click **YES** if you agree with the terms of the License Agreement.
   
   The Customer Information dialog displays.
4. At the Customer Information dialog, type your name, your organization’s name, and your MapMarker Plus serial number. The serial number is located on a sheet of stickers in the MapMarker Plus package.

To specify that you will be the only one using this installation, click the Only for me (user name) button.

To specify that others will be using this installation, click the Anyone who uses this computer button.

Your choice tells the installer where to place your shortcuts and user files. Choosing “only for me” will place the program shortcut in your private user profile. MapMarker will not be available to other users who log on to this computer using a different user name.

Click **NEXT**.

The Data Unlock Code dialog displays.

5. At the Data Unlock Code dialog, type your data unlocking code in the box. This code enables the installer to create a license file for your system that gives you access to the data you purchased. Please call (800) 552-2511 option 3, between 8 am and 7 pm, EST, Monday through Friday, and the MapMarker License Authorization desk will provide your unlocking code. Have your product serial number ready when you call. If you are evaluating this product, enter EVAL for the unlocking code.
If you have purchased a single state, or a six-state pack, you will be able to install those states after entering your unlocking code. If you would like to evaluate data for the entire United States, enter “EVAL” and an evaluation license will be created.

Click NEXT.

The Setup Type dialog displays.

6. Select either Complete or Custom. Choose Complete to install MapMarker Plus and all of its components. Select Custom to choose which program features and data to install. See Creating Data Subsets on page 28 for more information.

Note: To create a subset of your data to store on your hard drive select a Custom installation.

Click NEXT.

The Choose Installation Locations dialog displays.

7. At the Choose Installation Locations dialog, accept the default installation location for the MapMarker software and data, or click the appropriate Browse button to select a different location. The default location for the MapMarker software is c:\Program Files\MapInfo\MapMarker. The default location for the MapMarker data and license file is c:\Program Files\MapInfo\MapMarker\Data.
Next, indicate how MapMarker will be accessed. Select the “This installation is for a single user” check box to indicate a local installation. Select the “This installation will be shared with other users” check box to indicate that this installation of MapMarker will serve as a host and provide a group of users on the same network a common location for the software. Selecting this button ensures that users accessing MapMarker Plus from the shared machine will also be able to access the data.

Those who wish to access the shared installation will need to perform a client installation. See Client Installation on page 23 for more information.

When you select the option for a shared installation, a dialog will display, asking you if you would like to set up the client on this machine.

If you plan to use the MapMarker application on this machine, select Yes. If you are installing the software only for other users to access, select No.

Those who wish to install MapMarker as an RPC service, an OLE API server, or as a Geocoding server for intranet/Internet requests should choose the single user option. Click NEXT.
8. If you selected a Custom setup, the Custom Setup dialog displays. If you selected a Complete setup, the Custom Setup dialog does not appear. For a single-user installation, the installer will copy all the software and data to the locations specified in the file locations dialog. For a shared installation, only the application files, client setup program, and data will be installed.

Select which program features and data you want to install, and indicate whether you would like to install some, all, or none of the data to your local hard drive.

Click NEXT.

If you indicated that this installation would be used by others on a network, the Provide shared folder information dialog displays. Continue to step 9. If you indicated that this installation would be for a single user, proceed to step 10.

9. At the Provide shared folder information dialog, confirm or correct the existing path to the Address Dictionary and add additional paths if appropriate. Separate each path by a semicolon. Universal Naming Convention (UNC) paths are also accepted (e.g., \Discovery\mapmarker\data).

Click NEXT.

The Ready to Install the Program dialog displays.

10. At the Ready to Install the Program dialog, click Install.
11. The Installing MapMarker Plus dialog displays on your screen. It shows a status bar that indicates the progress of the installation.
When all files have been installed, the Setup Complete dialog displays.

12. At the Setup Complete dialog box, select the check box to view the readme file. Click Finish to complete the installation.
If you chose a shared installation and specified that you wanted to set up the client on this machine, the client installer will launch automatically when this installation is complete.

Shared Installation

A MapMarker installation can be shared among a group of users. The system administrator performs a complete installation of MapMarker (which includes the Address Dictionary) in a common network location. Once the installation of software and data is completed, the client runs SETUP.EXE from the MapMarker directory to set up the client machine to access the shared program.

The MapMarker installation creates a file called setup.ini in the program directory. This file contains important information for the system administrator, including the product serial number and path(s) to the access licences and Address Dictionary.

You can install all the data on a network directory (recommended) or access the data from a shared CD-ROM drive.

Client Installation

To set up a client workstation to access MapMarker from a shared installation on a network server, follow the steps in this section.

1. Browse to the location of the shared installation of MapMarker. (You may need to check with the System Administrator for the location.)
2. From the shared machine, run the setup program (SETUP.EXE) in the MapMarker program directory.
The Welcome dialog displays.

3. At the Welcome dialog, click **NEXT** to proceed.
The Ready to Install the Program dialog displays.

4. At the Ready to Install the Program dialog, click **NEXT**. The Setup Complete dialog displays when the installation is finished.

   All the components necessary to run MapMarker as a client are now installed. This includes the data, as well as MDAC 2.7 and the Microsoft Jet engine with their desktop ODBC drivers.

   The installer creates a MapMarker program icon on the client machine, places appropriate system files in the Windows system directory, and updates the registry.
Unlocking DPV

Customers who purchase DPV must obtain an unlocking code from MapInfo’s Customer Service department before they can use the feature. To obtain the unlocking code, customers must generate a system key on the computer where MapMarker is installed and provide the key to MapInfo Customer Service.

The DPV Unlocker application that comes with MapMarker generates the key and unlocks the DPV data once the unlocking code is entered into the application. The application is installed on your system when you install MapMarker and is available in the MapMarker directory. You will need to run the DPV Unlocker before you use DPV for the first time. You may also need to run the DPV Unlocker to re-enable DPV after a usage violation. See Usage Requirements on page 76 for more specific information.

To obtain the unlocking code:

1. In the DPV Unlocker application window, you will see a System Key displayed in the System Key box. Call MapInfo Customer Service at 1-800-552-2511 and supply the Customer Service Representative with the key. They will in turn provide you with an unlocking code.
2. Enter the unlocking code into the Unlocking Code box.
3. Click Unlock DPV.
4. Click Close to close the program.

MDAC 2.7 and MapMarker ODBC Support for Remote Databases

Microsoft Data Access Components 2.7 is required to use MapMarker with a SQL database. This component is installed automatically when you install MapMarker Plus 10.0 either on a standalone workstation or on a client machine. The MapMarker Plus installer and the client setup program both install the Jet 4.0 database engine for use with Microsoft Access. The Jet 4.0 package and the MDAC 2.7 installer together install a number of ODBC drivers for use with MapMarker.
Among them are the following supported drivers:

- SQL Server v. 2000.81.9001.00
- Microsoft Access driver (*.mdb) v. 4.00.5303.01

MapMarker also supports the following Oracle driver:

- Oracle 9.2.06

You must install the Oracle driver separately. To use the driver, you must have the Oracle 9i client installed.

## Installing MapMarker Server, OCX, and APIs

The MapMarker Server, OCX, and APIs are installed at the time you perform a full installation or upgrade MapMarker’s software and data.

### Installing MapMarker Server Manually

The MapMarker installer takes care of the MapMarker Server registration for you; however, if you need to install MapMarker as a Windows Service manually:

- At the DOS prompt, type `mm_serve -install`

Mm_serve.exe can be found in the MapMarker program directory.

### Registering the OCX Manually

If you need to register the OCX manually:

- At the DOS prompt, type `regsvr32 mapmarkr.ocx`

Regsvr32.exe is located in the System32 folder in your Windows directory. mapmarkr.ocx is found in `\Program Files\Common Files\MapInfo Shared\Common dlls\` on the target machine.

### Installing the MapMarker APIs

The files needed to use the MapMarker Application Program Interfaces (APIs) are installed when you choose a full installation or when you select the **DEVELOPER TOOLS** option in the Custom Setup dialog during a custom installation. The APIs include:

- OLE Automation API for developing OCX controls.
- RPC API for creating C client applications that call MapMarker Server.
- MapMarker GeoEngine API for creating stand-alone geocoding applications in C.

Following installation, the `geoeng` subdirectory, under the MapMarker directory contains header files and sample code. For more on these APIs, see the MapMarker Product guide.
Starting MapMarker

To start MapMarker:

- In Windows 2000/XP or NT 4.0, choose MapMarker from the Start menu or Shortcut menu.

Removing MapMarker From Your System

MapMarker installs a program that you can use to remove MapMarker from your system when necessary. Remove MapMarker by using the Add/Remove Programs utility from the Control Panel. MapMarker’s uninstall program removes the MapMarker program directories, groups, and icons, as well as any MapMarker files placed in the Windows\System directory. It also deletes the registry entries pertaining to MapMarker. Furthermore, if you have MapMarker as a shared installation, the uninstaller will remove the MapMarker Client program if it was installed.

Installation Considerations

The following topics discuss specific installation situations.

Users of MapX 4.x

MapMarker Plus 10.0 uses MapX as its engine for candidate visualization. The MapMarker installer installs a 4.5 version of MapX on your system. MapMarker does not overwrite MapX v. 3.x or 5.x installations, but it does replace some of the files in a MapX 4.0 installation with the MapX 4.5 versions.

If you remove MapMarker, the original MapX is left on your system. If, however, you remove your copy of MapX, the MapMarker version of MapX is also removed.

Multiple Versions of MapMarker

You can have more than one version of MapMarker installed on your system at a time as long as they are in separate directories. For example, you can install MapMarker Plus 10.0 on your system, but continue to geocode your tables with the Plus 9.x version. MapMarker Plus 10.0 is listed in the registry under its own version key under HKEY_LOCAL_MACHINE\Software\MapInfo\MapMarker\<version number>\.

Note: In references to the MapMarker registry keys, <version number> indicates the major-release version of MapMarker, e.g., 10.0, even if you are running one of the point releases of that version, e.g., 10.1, 10.2.
Windows Registry

The Registry values for MapMarker Plus are located under HKEY_LOCAL_MACHINE \SOFTWARE\MapInfo\MapMarker\ and in HKEY_CURRENT_USER\Software\MapInfo\MapMarker\<version number> after installation.

For users of the MapMarker APIs, the DatabasePath is listed under:
HKEY_LOCAL_MACHINE\Software\MapInfo\MapMarker\<version number>\System\

Note: In references to the MapMarker registry keys, <version number> indicates the major-release version of MapMarker, e.g., 10.0, even if you are running one of the point releases of that version, e.g., 10.1, 10.2.

For Developers of MapMarker Applications

MapMarker Plus 10.0 provides engine.dlls for versions 5, 6, 7, 8, and 9 (mm32v5.dll, mm32v6.dll, etc.) to provide backwards compatibility for custom applications. For example, to use your 7.x applications in MapMarker Plus 10.0, be sure that mm32v7.dll, mm32v8.dll, mm32v9.dll, and mm32v10.dll are accessible to your application.

Required Files for Using the MapMarker Geocoder Control (OCX)

The MapMarker Geocoder Control is installed on your system when you install or upgrade MapMarker Plus. The required files for using the control are mapmarkr.ocx, and mm32v10.dll. The files required for using the MapMarker geocoding control are located in the Program Files\Common Files\MapInfo Shared\Common dlls folder.

Creating Data Subsets

MapMarker’s performance increases if it accesses the data directly from your hard drive. However, you may not want the installer to copy the entire U.S. Address Dictionary to the hard drive. To save disk space, create a subset of the Address Dictionary that you routinely access and copy it from the CD-ROM.

You can create data subsets directly during the installation if you perform a Custom install. In the Custom Setup dialog, expand the Data files to see the data that you are installing. The custom installer default is to copy files to the local hard drive. You must choose the states that you want to remain on the CD by selecting “This feature will be installed to run from CD.”

If you opted to use the data from CD during the installation, you can rerun the MapMarker product installer and change your selections in the Custom Setup dialog, or you can copy the state files you need to create a subset of the Address Dictionary.

MapMarker’s U.S. Address Dictionary is organized by 2-digit ZIP Codes and includes state centroid and configuration files. For example, a complete data set for the 95xxx ZIP Code area in California contains:
State Centroid file:
- caz9.cen

Military Address Dictionary files:
- MX.*

2-digit ZIP Code file:
- ca95.adr

2-digit ZIP Code file:
- ca95.adx

Configuration files:
- usafin.*
- zipmastr.*
- zipmove.*
- geousa.*
- airport.*
- cityfin.idx
- cityline.*
- cityurb.cdb
- cityunz.*
- cityunz4.*
- cityzip.*
- finctrd.idx
- file.tbl
- sinames.*
- *.las
- *.los
- ecodes.dbf
- ecodes.cdx
- pointzips.dbf
- pointzips.cdx
- highrise.*

The configuration files have already been installed on your system in the location you specified at installation. You must copy the state centroid file and the appropriate .adr/.adx pair(s) to the same directory.

To create a data subset:

1. Copy the following files from the CD-ROM to your hard drive where the MapMarker configuration files are located. The default location is \mapmarker\data.
   - xxz9.cen
   - xxnn.adr
   - xxnn.adx

   where xx is the state abbreviation and nn equals the 2-digit ZIP Code files.

   The 2-digit ZIP Code .adr/.adx pairs vary from about 2 MB to 52 MB, while the state centroid files vary from 100 KB (Vermont) to about 8 MB (California).
2. Once you have moved the files, set the path in MapMarker to point to your data directory. To do this, start MapMarker and select **OPTIONS > SYSTEM PREFERENCES > DICTIONARY** and specify the path in the Address Dictionary box by typing it in or browsing to the location.

The Address Dictionary path can contain multiple directories separated by semicolons. MapMarker always searches your data path in the order given. For example:

```
c:\mapinfo\mapmarker\data\calif;d:\data
```

You may wish to include the path for the data CD-ROM as well so that MapMarker can match to addresses outside the subset when necessary.

---

### Moving Data and Program Files

Moving the MapMarker Program and Data files is not recommended. The product uninstaller will fail if it does not find the files and directories in their original location. The safest way to move the software to a new location is to remove the program and reinstall it in its intended target directory.

For a list of program files included in the MapMarker product, see **Appendix G: MapMarker Program Files** in the MapMarker Plus User Guide.
Now that you have been introduced to MapMarker, it's time to get to the business of geocoding your table. This chapter shows you how to run MapMarker. It begins with a discussion on preparing to geocode that can help you get the most out of running MapMarker right from the start.

In this chapter:

- **Before You Geocode** ................................................................. 32
- **Running MapMarker** ............................................................... 37
- **Identifying Address Columns** ................................................... 37
- **Geocoding Your Table Automatically** ................................. 40
- **Geocoding Your Table Interactively** ........................................ 41
- **Making the Most of Interactive Geocoding** .......................... 43
- **Displaying Geocoded Records in MapInfo** ............................ 44
- **Re-Geocoding a Table** ............................................................. 45
- **Setting Geocoding Preferences** .............................................. 45
- **Saving Settings to a Table** ....................................................... 50
Before You Geocode

MapMarker is simple to use. All you do is open your table, specify which columns contain address information, and then from the TABLE menu, click GEOCODE. MapMarker automatically matches your records to those in its Address Dictionary, determines if there is a match, assigns the geographic coordinates, and optionally creates a point.

Sort Before Geocoding

For fastest operation, we recommend that you sort your table by ZIP Code before running MapMarker. You do not need to scrub your data first since MapMarker’s sophisticated matching technology can read and successfully geocode a wide variety of addresses. MapMarker includes an interactive geocoding mode that lets you, not only correct your record to make a match, but have the edits permanently written to your base table.

Input Address Columns

MapMarker reads the addresses stored in your table and matches it against address records in the Address Dictionary. You must specify the columns that contain your addresses. The Select Input Columns dialog box is the first area where you specify that information. For street-level geocoding, have at least one street address column plus one column each for city and state or ZIP Code. For ZIP Code centroid matching, you do not need to specify a street address or city and state.

Output Address Columns

MapMarker can store longitude/latitude coordinates and geocoding result codes in your table. For MapInfo (.tab) or dBASE (.dbf) tables you can either prepare the table ahead of time by adding columns to store this information, or specify that MapMarker automatically creates the columns for you. For ODBC (remote) tables, you or your database administrator must add the columns before you run MapMarker.

Note: The longitude/latitude columns are required if you do not want MapMarker to create points automatically. A result code column is optional, but it is recommended.

If you want MapMarker to return the matching address for you to compare with your input address, you must provide columns for that information. For .tab and .dbf files you can either prepare your table ahead of time, or create the columns by clicking the MODIFY COLUMNS button on the Select input columns from your table or Select output columns from your table dialog boxes. For remote tables, add the columns before opening the file in MapMarker. If you store your input addresses in two fields, note that MapMarker only returns the matching address to a single field. ZIP Code and ZIP+4 are returned to two fields.

MapMarker can also return attributes from other tables during geocoding, such as demographic data, so prepare columns first or on the fly from the Select input columns from your table dialog box, as described above.
Output Column Type/Width Requirements

The table below lists the type and width requirements for output columns. Note certain columns are required when you are geocoding for CASS certification.

<table>
<thead>
<tr>
<th>Output Column</th>
<th>Type (Width)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firm*</td>
<td>Character(30)</td>
</tr>
<tr>
<td>Address*</td>
<td>Character(64)</td>
</tr>
<tr>
<td>Address2*</td>
<td>Character(64)</td>
</tr>
<tr>
<td>City*</td>
<td>Character(30)</td>
</tr>
<tr>
<td>State*</td>
<td>Character(2)</td>
</tr>
<tr>
<td>ZIP Code*</td>
<td>Character(5)</td>
</tr>
<tr>
<td>ZIP+4*</td>
<td>Character(4)</td>
</tr>
<tr>
<td>Urbanization*</td>
<td>Character(30)</td>
</tr>
<tr>
<td>Delivery Point*</td>
<td>Character(2)</td>
</tr>
<tr>
<td>Check Digit*</td>
<td>Character(1)</td>
</tr>
<tr>
<td>LACS*</td>
<td>Character(1)</td>
</tr>
<tr>
<td>PMB*</td>
<td>Character(4)</td>
</tr>
<tr>
<td>PMB range*</td>
<td>Character(8)</td>
</tr>
<tr>
<td>Longitude</td>
<td>Decimal(11,6) or Float</td>
</tr>
<tr>
<td>Latitude</td>
<td>Decimal(11,6) or Float</td>
</tr>
<tr>
<td>Result Code</td>
<td>Character(13)</td>
</tr>
<tr>
<td>Carrier Route</td>
<td>Character(4)</td>
</tr>
<tr>
<td>Census Block ID</td>
<td>Character(15)</td>
</tr>
<tr>
<td>DefaultFlag</td>
<td>Character(1)</td>
</tr>
<tr>
<td>EWS</td>
<td>Character(1)</td>
</tr>
<tr>
<td>RecordType</td>
<td>Character(1)</td>
</tr>
<tr>
<td>additional fields for attribute data</td>
<td>Character (depends on the data)</td>
</tr>
</tbody>
</table>

* CASS required field
Census Block IDs

MapMarker can return Census Block IDs as output, or partial Census information, depending on the quality of the match. The complete Census Block ID is a 15-character code and is made up of the following:

SSCCCTTTTTTTGBB(B)

where S=State FIPS (Federal Information Processing Standard) code (two characters)
C=County FIPS code (three characters)
T=Census Tract (six characters)
G=Census Block Group (one character)
B=Census Block (typically two characters, sometimes 3)

MapMarker can return the full 15-character Census Block ID only for a geocoded record with an S5 result code. The S5 result code indicates that the point is located at a street address position. This code represents the highest accuracy available for a geocoded point.

Some S codes and Z codes may return partial census information if ZIP+4 information is supplied. Others may not return any census information because certain information is missing. The following table shows what census information is returned, if any, for specific result codes.

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Census Block ID Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S5</td>
<td>Returns complete 15-character Census Block ID. S5 results are matched to a street address location.</td>
</tr>
<tr>
<td>S4</td>
<td>None. A house number is required for Census Block IDs. S4 results are matched to a shape point path.</td>
</tr>
<tr>
<td>S3</td>
<td>May return a 12-character Census Block Group number. S3 results are matched to ZIP+4 locations, which may cover more than one Census Block.</td>
</tr>
<tr>
<td>S2</td>
<td>May return partial census information if ZIP+4 is supplied. S2 results are matched to ZIP+2 locations.</td>
</tr>
<tr>
<td>S1</td>
<td>May return partial census information if ZIP+4 is supplied. S1 results are matched to ZIP centroid locations.</td>
</tr>
<tr>
<td>Z3</td>
<td>May return partial census information if ZIP+4 is supplied. Z3 results are matched to ZIP+4 locations.</td>
</tr>
<tr>
<td>Z2</td>
<td>May return partial census information if ZIP+4 is supplied. Z2 results are matched to ZIP+2 locations.</td>
</tr>
<tr>
<td>Z1</td>
<td>May return partial census information if ZIP+4 is supplied. Z1 results are matched to ZIP centroid locations.</td>
</tr>
</tbody>
</table>

For more information on result codes, see **Chapter 5: Result Codes**.
Output Field Size and Column Names

It is possible that the recommended length of your output fields for Street and Firm may not be long enough to contain the complete output information. This is due to the increased output information such as firm names, apartment numbers, and suites, that MapMarker can return. When this happens, MapMarker truncates the information in the respective output field.

In the Input Column and Output Column dialog boxes, if a field name is too long to be seen in its entirety in the list box, hold the cursor over the field name and a popup box with the full name of the field is displayed.

Mixed Case Street Output

This optional feature adds the ability for output street addresses to have mixed-case. The capability is supplied in the MapMarker GUI only. It is a geocoding preference located on the Geocode tab of the Geocode dialog box. When the mixed-case check box is selected, the output street is in mixed-case for a match. The input street column remains unchanged.

An output address without a house number remains in uppercase. This is to accommodate cases such as PO BOX 233. Streets such as McFly are handled so that the letter following Mc is capitalized. However, there is no way handle cases like Macintosh because it cannot be differentiated from cases like Macoun.

Setting Startup Preferences

MapMarker provides startup preferences to get you through the setup phase of geocoding with ease. These startup preferences are available on the System Preferences dialog box. To display the System Preferences dialog box, click SYSTEM PREFERENCES on the OPTIONS menu, and then click the STARTUP tab.
You can use any and all preferences available in this dialog box. For the file open choices, if you select either Show MapInfo/dBASE files or Show for ODBC tables, the Open dialog box displays automatically when you start MapMarker. If you have both options selected, MapMarker prompts you with another dialog box at startup to choose which type of table you want to open.

You may also control which of the setup dialog boxes, if any, display on startup. Choose from:

- Show Select Column Dialog Box On Open (selected by default)
- Select Attribute Files After Selecting Columns
- Show Geocode Dialog Box After Selecting Columns (selected by default)

By selecting all three, when you start MapMarker, it walks you through the geocode and attribution setup phase. The settings are saved to the table as metadata and used as new defaults when you open the table again.

Other system preferences are discussed in Setting System Preferences in Appendix B on page 370.
Running MapMarker

MapMarker geocodes your records to street level or ZIP Code centroid either automatically (making a match without your involvement) or interactively (allowing you to decide the match from a list of suggestions). In brief, the geocoding process follows these steps:

1. Open the table to be geocoded and identify the columns that contain addresses (see Identifying Address Columns on page 37).
2. Geocode your records automatically to match as many records as possible (see Geocoding Your Table Automatically on page 40).
3. Geocode remaining unmatched records interactively by choosing from a list of candidates (see Geocoding Your Table Interactively on page 41).
4. Display geocoded records in MapInfo mapping software (see Displaying Geocoded Records in MapInfo on page 44).
5. Analyze the result codes (see Result Codes on page 61).

Identifying Address Columns

The first step in geocoding with MapMarker is identifying the columns in your database that contain address information that can be used to attempt a match.

Note: This procedure applies to opening MapInfo .tab and .dbf files. For instructions on opening remote tables, see Chapter 7: Geocoding Remote Tables.

1. Start MapMarker from the Windows Start menu or Shortcut menu.

   If MapMarker is already running, choose FILE > OPEN TABLE, click or click the secondary mouse button to display a shortcut menu. The OPEN dialog box displays.

2. Choose the .tab or .dbf table you wish to geocode. Click OPEN.

3. Click OK. A progress bar indicates your table is being updated. Following that, a browser window containing your table and the Select input columns from your table dialog box displays.
4. Choose the columns that contain the appropriate address components. If your table has separate columns for ZIP Code and ZIP+4 information, identify each column in the ZIP Code and ZIP+4 boxes. Notice that MapMarker attempts to choose the address columns for you. It can make some determinations for fields, but if this is the first time this table has been opened in MapMarker, be sure to check these columns. Once you have geocoded the table, the column settings are saved to the table as metadata and used for future geocoding sessions.

5. Click the NEXT button. The Select output columns from your table dialog box displays.
6. Identify columns in your table where MapMarker will store the result codes, coordinate information, and output addresses from the Address Dictionary.

If you do not want MapMarker to create points for your table, clear the check box marked **CREATE POINTS IN MAPINFO TABLE**. In that case, you must choose longitude and latitude columns in order to proceed with geocoding.

Specify the columns to contain output information from the Address Dictionary for each matched record. If you are geocoding for CASS certification, all fields in this group with the exception of Census and Carrier Route are required.

**Note:** Use care when choosing the same columns for output and input addresses. MapMarker will overwrite your data with the output address.

MapMarker only returns the parsed street name to a single field.

Identify a census column if you want MapMarker to return the Census Block code for the matched address. MapMarker returns the code that represents the state and county FIPS code, census tract, and tabulation block, depending on the match type.

7. Once you have identified all your columns, click **Finish**. You are automatically returned to the MapMarker menu or to the **GEOCODE** dialog box.
Geocoding Your Table Automatically

To geocode automatically:

1. If the GEOCODE dialog box is not automatically displayed upon leaving the SELECT OUTPUT dialog box, do one of the following:
   - Choose TABLE GEOCODE from the MapMarker menu.
   - Click the button.
   - Click the secondary mouse button, and choose Geocode from the pop-up menu.

2. Set up the geocoding parameters here. Choose the appropriate geocoding precision: street level or ZIP Code centroid. Specify how much of the table to geocode. By default, MapMarker automatically geocodes the unmatched rows.

   For previously geocoded tables, you can also try to improve a match for a record by geocoding rows that have result codes below the desired level.

3. Set additional match settings by selecting the appropriate check boxes in the other GEOCODE dialog boxes. For an explanation of these settings, see Setting Geocoding Preferences on page 45.

4. Click GEOCODE to begin geocoding. The AUTOMATIC GEOCODE dialog box is displayed showing a progress bar that updates the match count every 25 records.

5. At completion of the geocoding run, the CANCEL button changes to DONE. Click DONE to leave the summary dialog box. MapMarker opens the Log file if you selected AUTOMATICALLY OPEN LOG FILE AFTER GEOCODING in the Log File dialog box.

   The Log file contains a history of the geocoding pass, including preferences, the number of matched, unmatched and previously matched records, and the number of matched records by result code.

   Information from subsequent passes is appended to the top of the Log file. The Log file size limit is 1 MB.

6. To view the SUMMARY dialog box again, choose VIEW SUMMARY. Note that only the summary from the most recent geocoding pass for the active table is available from this menu.

7. Optionally, to create a summary of the geocoding results for a table, you can choose TABLE TABLE SUMMARY. Specify the column containing the georesult and save the information as a text file.
Note: The summary is for the active table (the frontmost browser). If you run multiple tables in a single session, you will have addresses open for each table. To obtain the summary information for the desired table, click on the Browser title bar to make it active and choose View -> Summary.

To view information about previous geocoding passes, view the Log file by choosing View->Log File.

Geocoding Your Table Interactively

MapMarker allows you to geocode records interactively by choosing from a list of candidates. Using interactive geocoding, you can expand the search for a match, map potential matches, move back up the table to an earlier record, and correct your records directly from the MapMarker dialog box.

Note: Interactive geocoding is not completely interactive. MapMarker attempts to geocode the row first. Only when it fails to find a single close match is the record displayed in the Interactive dialog box.

1. Do one of the following:
   - Choose Table->Geocode from the MapMarker menu.
   - Click the button.
   - Click the secondary mouse button, and choose Geocode from the pop-up menu.

2. In the Geocode Precision group, choose Street Level and set Method to Interactive.

3. Choose the range and type of rows to geocode.

4. Click Geocode. The Interactive Geocode dialog box is displayed showing the first unmatched record and a list of suggested matches. The matches contain a result code category, address number, street name, city, state, and ZIP Code.
The input address shows the row number for the record by default. If you wish to display another field from the input table choose it from the drop-down list.

5. Choose the **GEOCODE** button if you agree that the highlighted suggested match is the correct match. MapMarker assigns coordinates from that row and moves on to display the next unmatched record. To choose a different candidate in the list, move the highlight with your cursor to your choice and choose **GEOCODE**.

6. Choose **IGNORE** if MapMarker offers no match suggestions or if you do not agree with the choices in the match list.

When you click **IGNORE**, a dialog box is displayed asking if you want to mark the record as “Non-geocodable.” This means that you do not want MapMarker to attempt to match this record during future geocoding passes. This is useful when you know the record does not geocode and you do not want it to display again in future geocoding passes. You have several choices in this dialog box:

- Click **YES** to have MapMarker mark the record with the result code NG (non-geocodable). This dialog box is displayed each time you click **IGNORE**.
- Click **YES TO ALL** if you want to mark all future “Ignore” records in this geocoding session as NG. The dialog box is not displayed again during the session.
- Click **NO** if you do not wish to mark the record as NG. The dialog box is displayed each time you click **IGNORE** on a record.
- Click **NO TO ALL** if you do not want to mark any “Ignore” records as NG. The dialog box is not displayed again during this session.

7. To expand the search area, select the **EXPAND SEARCH** check box and specify a radius of up to 99 miles. Select the check box labelled **LIMIT TO CURRENT STATE** if desired. This feature is helpful to find a match when the input address contains limited or inaccurate city or ZIP Code information.

8. To change the matching conditions for a record, click the **OPTIONS** button. The tabs for **MATCH RESTRICTIONS**, **fallback**, and **MULTIPLE MATCH** are displayed. Make the necessary changes to increase the potential for a match. Click **SEARCH** to search the Address Dictionary using the new matching conditions.

9. Edit the record by typing in new text that may increase the likelihood of a match. Click **SEARCH** to bring up a new list of suggestions that may better match the edited record. Additionally, incorporate the edits into your table by selecting the **WRITE SELECTED ADDRESS BELOW TO DATABASE ON GEOCODE** check box. Choose the match from the list and click **GEOCODE**. MapMarker geocodes the record and updates your table with your edits.

10. To see an expanded list of ranges, including apartment numbers, office suites, etc., choose **RANGES**. A detailed list of address ranges for the highlighted street is displayed. Click on the **RANGES** button again to close the list of ranges.

11. Choose the **BACK** button to move backward in the table to redisplay a record.

12. Choose the **MAP** button to view the match candidates in a map window. The match candidates are displayed with color-coded symbols that indicate their match potential to the input address.

   The map feature is available if you have set the location for street data in the Maps tab in the **SYSTEM PREFERENCES** dialog box. See **Viewing Match Candidates on a Map on page 57**, for more information.

13. Click **CLOSE**. A **GEOCODING SUMMARY** is displayed showing the results of the session.
14. Click **DONE**. To view the summary again, choose **VIEW>SUMMARY**.

15. Optionally, to create a summary of the geocoding results for a table, you can choose **TABLE>TABLE SUMMARY**. Specify the column containing the georesult and save the information as a text file.

### Making the Most of Interactive Geocoding

Interactive geocoding allows you to control the matching process because you can accept or ignore a potential match from the list of suggestions that MapMarker generates for the record.

#### Change Matching Conditions

MapMarker follows the settings in the **GEOCODE** dialog box or responds to your changed settings as you progress through the interactive pass. To increase the list of suggested matches, relax street name. You can do this at the beginning of an interactive pass or change the settings for each record by clicking the **OPTIONS** button on the Interactive Geocode dialog box.

You can also return to a previous record in the table with the **BACK** button and change settings as needed. This feature can help you maximize the potential for a match.

#### Map the Match Candidates

To help you choose the best match for your record, display the candidates on a map via the **MAP** button in the **INTERACTIVE** dialog box. This requires that you have street data on your system and you specify its location in the **SYSTEM PREFERENCES>MAPS** tab. See **Viewing Match Candidates on a Map** on page 57 for a complete discussion of visualizing match candidates.

#### Expand Your Search

Once at the **INTERACTIVE GEOCODE** dialog box, you can further increase the likelihood of a match by expanding the search area in which MapMarker checks for a match. Select the **EXPAND SEARCH** check box and set the radius in miles (up to 99) around which your record lies. You can contain the search to the state by selecting the **LIMIT TO CURRENT STATE** check box. MapMarker checks all finance areas whose centroids lie within the radius.
Edit Your Records

Interactive geocoding also allows you to correct your record to improve the possibility of a match. If you choose, your table can be updated to reflect the newly edited record, all from within the INTERACTIVE GEOCODE dialog box.

By manually correcting the fields to put the firm, street, city, and state into their proper locations, the record is more logical and MapMarker can make better suggestions on possible matches. After editing the record, click the SEARCH button to display a new list of match candidates.

To write these changes to the base table, select the WRITE SELECTED ADDRESS BELOW TO DATABASE ON GEOCODE check box and click Geocode to confirm the match.

Note: Your edits will overwrite the address data in the respective fields, so be sure not to alter the fields containing the correct address data.

Displaying Geocoded Records in MapInfo

After MapMarker geocodes your records, you can display them in MapInfo Professional or MapX to analyze them geographically.

1. Close your table in MapMarker and open it in MapInfo Professional or add it to your MapX geoset.
   The points are displayed with color-coded symbols that represent the level of geocoding precision for the record, as described below.
   - **Green stars**: all records geocoded to a street address (S5 and M5 result codes)
   - **Blue stars**: All records geocoded to a shape path centroid (S4, M4 result codes)
   - **Yellow stars**: All records geocoded to a ZIP+4 centroid (S3, M3 and Z3 result codes)
   - **Cyan stars**: All records geocoded to a ZIP+2 centroid (S2, M2 and Z2 result codes)
   - **Red stars**: All records geocoded to a ZIP Code centroid, including highway exits (S1, M1 and Z1 result codes)
   - **Purple stars**: All records geocoded to point zip locations (S6, M6 and Z6 result codes)
   - **Black stars**: All records geocoded to intersections (Mx and Sx result codes)

   See Result Codes in Chapter 5 on page 61 for an explanation of the types of result codes.

2. Add other map layers such as MapMarker Streets or ZIP Code boundaries to give your points a geographic reference.

You are now ready to analyze your data geographically using the power of MapInfo mapping software.

When displaying your geocoded points over street data, use MapMarker Streets, StreetInfo, or StreetPro display (products that use TIGER 2000 street geometry). When geocoding with MapMarker Plus, be sure to display over enhanced TIGER 2000 street geometry, such as MapMarker Plus Streets or StreetPro Enhanced or Display. This preserves the positional accuracy of the points with respect to the street geometry. MapMarker Streets is included in the MapMarker standard product. MapMarker Plus Streets ships with MapMarker Plus.
Re-Geocoding a Table

You may often find that once a table is geocoded, records become updated or new ones added and you need to geocode again. New or previously ungeocoded records (those that do not have result codes) can simply be re-geocoded. By default MapMarker automatically geocodes only unmatched records. This means that for a large database, previously matched records are not re-matched, thus saving time.

Records that already contain coordinates, but are incorrect due to a change in address for example, must be identified and the result code deleted. MapMarker treats any record without a result code as an unmatched record and attempts to find a match. MapMarker also re-geocodes records with an N or ND codes.

If your table was geocoded previously with MapInfo Professional, keep in mind that MapMarker only replaces the points for matches it makes. It does not touch the points for records it cannot match.

Setting Geocoding Preferences

This section discusses geocoding matching options that are available from the tabs in the Geocode dialog box. These preferences affect the conditions under which MapMarker attempts to match a record. This includes automatic or interactive geocoding, exact or relaxed matching settings, fallback options, how to handle multiple matches, Log file setup, and CASS certification settings.

Changing match settings can affect the hit rate of successfully geocoded records. It can also affect the time in which MapMarker takes to geocode a table, as well as the precision with which the geocoded record is displayed on a map. It is important to keep in mind your intended use of geocoded data when setting these conditions. These settings are set for individual tables and the information is stored in the table’s metadata.

Accept or change any of the matching conditions to fit your needs. By relaxing some parameters, MapMarker is, in essence, expanding the search area or criteria used to match the record.

To set geocoding preferences for a table:

1. On the TABLE menu, click GEOCODE. The Geocode dialog box displays.
2. Choose the settings you want from any of the six geocoding preferences tabs: Geocode, Log File, Match Restrictions, Fallback, Multiple Match, or CASS.
3. Click APPLY to set the changes or GEOCODE to proceed directly to processing.

Geocode Preferences

The geocoding precision is how close a match is to the true location of the record you want to geocode. MapMarker provides for two types of precision: Street Level and ZIP Centroid.
Street Level precision means that MapMarker attempts to geocode all records to street address, but some matches may end up at a less precise location such as a ZIP centroid (ZIP+4, ZIP+2, or ZIP Code) or shape path.

ZIP Centroid precision means that MapMarker attempts to match a record to the most precise ZIP Code it finds. Keep in mind that a match to a ZIP+4 centroid is nearly as precise as a match to a street address. The added bonus of ZIP Centroid matching is the speed of the operation; MapMarker processes records quicker if it only has to match on the ZIP Code.

The disadvantage of ZIP Code matching is that MapMarker only examines the ZIP Code field. If your records only contain 5-digit ZIP Codes, MapMarker can do no better than match to 5-digit ZIP Code centroids. On the other hand, if you use street-level precision, MapMarker looks at both the street name and ZIP Code fields and attempts to return street-level coordinates and optionally fall back to ZIP Code coordinates, in many cases to ZIP+4 centroid.

The Mixed Letter Case feature adds the ability for output street addresses to have mixed-case. When this option is selected, the output street is in mixed-case for a match. The input street column remains unchanged.

An output address without a house number remains in uppercase. This is to accommodate cases such as PO BOX 233.

Streets such as McFly are handled so that the letter following Mc is capitalized. However, there is no way handle cases like MacIntosh because it cannot be differentiated from cases like Macoun.

The geocoding method defines the run type for geocoding operation. MapMarker processes records automatically or interactively (where you must choose the match from a list of candidates).

For automatic geocoding, the operation continues until all records are processed or the operation is cancelled.

For interactive geocoding, MapMarker geocodes records automatically, but stops on the record for which it cannot make a match and waits for a response.
Log File Preferences

The Log file is a text file that MapMarker generates during geocoding that contains setup and processing information about the geocoding session. It contains the results of the session, including number of matches, non-matches, previously matched records, and the number of matched records by result code. MapMarker appends new information to the top of the Log file for each geocoding pass. The Log file is named mapmarkr.log. The Log file is written to the MapMarker directory. The maximum file size is 1 MB.

To specify a new Log file, select the SPECIFY LOG FILE NAME check box in the Log File tab of the GEOCODE dialog box. You may enter a path as well as a name for the Log file (e.g., c:\temp\Us_addr.log). If the path specified is invalid, MapMarker creates MapMarker.log in the MapMarker program directory. A message is displayed stating that it could not create the Log file as specified.

In Windows 98/2000/XP, you must close the Log file from a previous session before you can open the table it refers to in MapMarker. If you do not, a message that the table may not have read/write access is displayed. Close the table in the text editor in order to proceed.

In addition to creating a log file with the results of the geocoding operation, you can generate a table summary that shows the results of the geocoding. To do so, choose TABLE>TABLE SUMMARY and specify the column in your table that contains the georesults of the active table. The SAVE AS dialog box appears allowing you to save this information as a text file.

Match Restrictions Preferences

The conditions in the MATCH RESTRICTION tab specify how precisely MapMarker attempts to match your record. By default MapMarker requires an exact match on House Number, but not for Street Name and ZIP Code—conditions that strike the best balance among hit rate, performance, and minimizing erroneous matches.
When the street name box is cleared, MapMarker considers similar street names. When the ZIP Code check box is cleared, it searches surrounding ZIP Codes for a match. When ZIP Code is selected, MapMarker only geocodes records that have a ZIP Code.

Other conditions of matching that you can set here are accepting only street level matches, geocoding to street intersections, and clearing existing output values when MapMarker does not match a record. By accepting only street level matches, MapMarker does not place your records at a ZIP Code centroid.

For street intersections MapMarker can read input street addresses that contain an “&&” between Street1 and Street2 in the same field. Note that this type of matching affects MapMarker’s geocoding speed since it requires additional processing.

You can control whether MapMarker blanks out existing output values for a record or keeps the information intact if it cannot make a match.

For example, your table may contain longitude and latitude values obtained from another source, and you want to maintain that information if MapMarker cannot match the record and return new coordinates. In the MATCH RESTRICTIONS tab clear the CLEAR OUTPUT COLUMNS IF NO MATCH check box before geocoding. The check box is selected by default, that is, MapMarker blanks out the existing values for non-matches.

If you wish to change the matching conditions, clear or select the appropriate check box in the group. For instance, clear the House Number check box and select ZIP Code to allow a close house number as a viable match and have MapMarker search for matches in the ZIP Code contained in your address.

Note: If you choose to relax the match on Street Name, MapMarker’s performance is affected because it has additional street match candidates to consider.

Fallback Preferences

If MapMarker cannot match a record to street level, it matches to the ZIP Code centroid, provided you have selected the Fallback to ZIP Code Centroid check box in the Fallback tab. You can further define the fallback to a particular level of ZIP Code as well. For example, if you only accept a ZIP+4 centroid as a fallback match, choose the last item (ZIP+4 Centroid only). The default behavior is for MapMarker not to fall back to ZIP Centroid.
Multiple Match Preferences

If MapMarker determines that more than one match candidate exists for a record, it follows the instructions set in the Multiple Match tab. Multiple match candidates are potentially strong matches, but none of them stands out enough from the others to be considered the definitive match. See the table below for an explanation of the multiple match options.

<table>
<thead>
<tr>
<th>Multiple Match Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do Not Match</td>
<td>Click this option to not allow a match to be made for the record if multiple matches occur.</td>
</tr>
<tr>
<td>Accept First</td>
<td>Click this option to accept the first candidate for the match.</td>
</tr>
<tr>
<td>Pick Street Address over ZIP+4</td>
<td>Click this option to accept the candidate with the most accurate street level match.</td>
</tr>
<tr>
<td>Pick ZIP+4 over Street Address</td>
<td>Click this option to accept the candidate with the most accurate ZIP+4 level match.</td>
</tr>
</tbody>
</table>
CASS Certification Preferences

MapMarker Plus can standardize your mailing addresses and print a report describing the success of the operation if you plan to qualify your table for bulk mailing discounts from the U.S. Postal Service. This process is known as Coding Accuracy Support System (CASS) certification.

Because the address standardization requirements for CASS certification are very rigid, you cannot control the geocoding settings you normally have when geocoding with MapMarker. For example, you cannot geocode to a user dictionary when you are CASS geocoding. The options in the geocoding preferences tabs are unavailable when CASS geocoding is in effect.

To set up for a CASS geocoding pass, in the CASS tab select the **OVERRIDE EXISTING GECODING SETTINGS WITH CASS DEFAULTS** check box. If you wish to qualify your table for CASS certification, you must print a report USPS Form 3553. The default setting is for MapMarker to geocode without using CASS settings.

Use EWS File setting is a feature that includes the USPS-supplied EWS file to be used during the geocode. The EWS file is a file produced by the USPS to report addresses not included in the regular monthly update. The use of this file is optional and is included here to enable MapInfo to stay in compliance with current CASS requirements.

Saving Settings to a Table

MapMarker saves the settings you choose for geocoding and stores them as metadata in your table. These settings include input and output columns, geocoding preferences such as match restrictions, fallback settings and CASS options, and system preferences for startup, datum and address dictionaries. The next time you open the table in MapMarker, you can proceed directly to geocoding. To view the settings, open the .TAB file in a text editor and look under the section called Metadata.
Match Settings and Strategies

In this chapter:

- MapMarker’s Geocoding Model .................................................. 52
- Match Settings ........................................................................ 53
- Choosing Match Strategies ....................................................... 53
- Picking Your Strategy: An Example .......................................... 54
- Impact of Relaxing Match Conditions ....................................... 56
- Viewing Match Candidates on a Map ....................................... 57
- A Closer Look at Multiple Matches .......................................... 59
MapMarker’s Geocoding Model

MapMarker is based on a model of relative matching that is governed by a set of weights that scores each portion of the address against candidate records (possible matches) in the Address Dictionary. The resulting scores are summed and the candidate’s total score is used to determine the best match or matches. An exact match is made when there is a candidate that scores well above other candidates. If there is no clear best match, then the record is not geocoded. You can, however, attempt to match those records later during interactive geocoding where you can pick what you consider the best match from a list of candidates.

In addition, the matching routine uses the geocoding preferences that are set for the table to determine whether certain matching conditions are required or relaxed. For example, MapMarker’s default preferences include relaxing a match on street name and ZIP Code, but requiring a match on the house number. This gives the best return of hit rate with the fewest erroneous matches (false positives) and the best performance.

Geocoding Trade-offs

With a relative matching system such as MapMarker’s, there are trade-offs that must be considered in light of how you use the geocoded data.

Consider questions like the following when setting up a geocoding session:

- What level of matching accuracy are you looking for (unique address match, close match)?
- What level of geographic accuracy do you need for your geocoded points (street level, ZIP Code centroid)?
- Is your goal to geocode as many records as possible?

The answers to these questions are driven by how you intend to use the geocoded records. For example, perhaps you are determining the location of a new retail store and need to know the distribution of current and potential customers. In this case, you want to geocode as many of these customers as possible and do not need an exact street address match for each one. Geocoding to ZIP+4 centroid or ZIP Code centroid is fine for your analysis.

On the other hand, if you, as a utility service coordinator, need to know where your customers are in relation to neighborhood gas lines, the positional accuracy of each customer is of critical importance to you. Geocoding to street level with strict matching preferences is your best strategy.
Match Settings

Under the Geocode dialog box are tab dialog boxes that allow you to set the Match Restrictions, Fallback, and Multiple Match settings for a geocoding pass. Here you set MapMarker to be as strict or relaxed as you need by selecting or clearing the appropriate check box. The default settings, which give you the best compromise among match rate, accuracy, and performance, are:

- Require an exact match for house number only (set in the Match Restrictions dialog box).
- Do not fall back to ZIP Code centroid matching (Fallback dialog box items cleared).
- Do not match if there are multiple match candidates (Multiple Match dialog box items unchecked).

The strictest matching conditions require an exact match on house number, street name, and ZIP Code and no fallback to ZIP Code centroids. MapMarker, in essence, is looking for an exact street address match within the ZIP Code named in your input address.

Relaxing the conditions tells MapMarker to broaden the area in which it searches for a match. For example, by relaxing the ZIP Code, MapMarker searches for candidates outside the ZIP Code but within the city of your input address.

To maximize the potential for a match, tell MapMarker what to do if it does not find a match. These settings include Fallback to ZIP Code centroid matching and match to the first of multiple matches.

For P.O. Box and rural route addresses, MapMarker automatically geocodes them to a ZIP Code centroid, as required by CASS standards.

Each of the settings above affect the match rate, performance, and the positional accuracy of the geocoded point. You must decide the best settings for your needs.

Choosing Match Strategies

Now let’s look at some matching strategies, including:

- Striking a Balance Between Match Rate and False Positives on page 53 (erroneous matches)
- Maximizing the Match Rate on page 54
- Maximizing Geographic Accuracy on page 54

You can choose one method or use them in combination.

Striking a Balance Between Match Rate and False Positives

This strategy is best used when you want to geocode as many records as possible automatically, but want to minimize the number of erroneous matches (false positives). You would rather have those records remain ungeocoded so that you can geocode them interactively later. It is likely that you want to analyze your data at a local level where the location of each record on a map is important.
False positives are matches MapMarker makes when it finds a street that sounds like the input street; or finds the same street in another town (if ZIP Code is relaxed), or finds the street but with a different house number (if House Number is relaxed). MapMarker, however, contains a sophisticated matcher that usually finds the correct address.

MapMarker’s default settings reflect this strategy—exact match is required for house number, but is relaxed for street name and ZIP Code. Post Office box and rural route addresses are automatically geocoded to the ZIP Code centroid. You do not need to set any conditions specific to these addresses.

Maximizing the Match Rate

To generate the highest match rate possible, relax the match restrictions on house number, ZIP Code, and street names in the Match Restrictions tab by clearing those check boxes. In addition, select Accept First (Multiple Match tab) and Fallback to ZIP Centroid (Fallback tab).

While this scenario likely yields false positives, this is the best match solution when you have large databases that need to be geocoded. You should decide if the percentage of false positives affects your analysis.

Note: You can evaluate false positives if you have MapMarker return output street, city, state, and ZIP Code values. Look for false positives by comparing the input address with the output address.

To further decrease the number of false positives without sacrificing hit rate, analyze the result codes after a geocoding session and re-geocode them interactively. More on result codes is found in Result Codes in Chapter 5 on page 61.

Maximizing Geographic Accuracy

If positional accuracy of the geocoded records is the most important aspect for you, choose the third strategy in which MapMarker returns the lowest number of erroneous matches. To do this, you must require MapMarker to find exact matches for all portions of the address—house number, street name, city, and ZIP Code. Also, do not accept the first of multiple matches or fall back to ZIP centroid. These records remain ungeocoded.

Choose this scenario when your analysis requires precise locations for your records. To increase the number of matches, your next step might be to use interactive geocoding. Interactive geocoding is discussed in Making the Most of Interactive Geocoding on page 43.

Picking Your Strategy: An Example

The following scenario outlines a three-pass geocoding session based on the first strategy: the user is looking to generate the highest match rate for a large database while minimizing false positives. The goal during the session is to set the conditions for that strategy in the first pass, then follow with an interactive session to pick up additional matches, and finally, a third pass to ZIP Code centroids to geocode any remaining records.
First Pass

Geocode the table automatically with the settings that generate the highest match rate with the fewest false positives:

- Require exact match on house number, relax street name, and ZIP Code (default settings). For even stricter matching, require exact match on street name and ZIP Code as well.
- Do not accept first of multiple matches.
- Do not fall back to ZIP Code centroid matching.

All records geocoded during this pass are exact or close matches that spot at the street level. Depending on the cleanliness of your data, the majority of your table is processed during this pass. This pass generates the least number of false positive matches.

Second Pass

For this pass, geocode all unmatched rows interactively with the following settings:

- Relax the settings for house number, street name, and ZIP Code.
- Do not accept first of multiple matches.
- Do not fall back to ZIP Code centroid matching.
- Edit the input address where necessary to increase the likelihood of a match.

Each match in this pass is still at street level accuracy. Since you are in interactive mode, you control every match, thus, minimizing the number of false positives.

You can also change the match settings (Options button), expand the Search, map the match candidates, and edit the record to find additional matches.

Third Pass

Perform a final geocoding pass on unmatched rows by geocoding to ZIP Centroid. Set the geocoding precision in the Geocode dialog box to To ZIP Centroid.

This pass picks up virtually all remaining records and geocodes them to ZIP Code centroid accuracy. If your data includes ZIP+4 information, some of these ZIP Code centroid matches could spot at the street level in as nearly the same location as unique and close street matches from the first pass.

One Pass Settings

If you do not want to geocode your table interactively, combine the first and third passes using the following settings in the Options dialog box:

- Require exact match on house number, relax street name, and ZIP Code.
- Do not accept first of multiple matches.
- Choose Fallback to ZIP Centroids.
Impact of Relaxing Match Conditions

Consider the following when you change the settings in the Geocode dialog box.

Relaxing ZIP Codes

When ZIP Codes are relaxed, MapMarker searches a wider area for a match. While this results in slower performance, the match rate is higher because MapMarker does not need to match exactly when it compares match candidates.

Relaxing City Name

When city names are relaxed, MapMarker searches on the street address matched to the particular ZIP Code, and considers other cities that do not match the city name, but do match the ZIP Code.

and Relaxing Street Name

MapMarker looks at all candidates with names that sound like the input address. This slows down MapMarker’s performance. On the plus side, since more candidates are examined, the match rate increases. If your table is indexed, the time difference between performance and match rate is reduced.

Relaxing House Number

Performance is not significantly affected when the house number setting is relaxed. It does, however, affect the type of match if the candidate address corresponds to a TIGER segment that does not contain any ranges. The type of match can also be affected when the house number range for a candidate does not contain the input house number.

Matching to Street Intersections

If you select the option Match to Street Intersections a noticeable a drop in geocoding speed occurs since MapMarker is now comparing two street names for every record. Choose this option only if your table contains a large number of street intersection records. Street1 and Street2 must be contained in the same field separated by an “&&” for MapMarker to recognize it as a street intersection.

Because of the increased processing time required for intersection matching, we recommend that you geocode your table without intersection matching for the first pass. Then, with intersection matching turned on, geocode the unmatched records during the second pass.

Note: If your data contains intersections separated by a single ampersand (&), you may receive unexpected results. We strongly recommend updating your data to contain double ampersands (&&) for street intersection records.
Geocoding to ZIP Centroids versus Falling Back to ZIP Centroid

When MapMarker matches to ZIP centroid it only looks at the ZIP Code in the input address. If your data has a 5-digit ZIP Code, MapMarker can only match to the 5-digit ZIP centroid.

If, on the other hand, you geocoded the same record at street level with a fallback to ZIP centroid, MapMarker takes into account both the 5-digit ZIP Code and street name and is likely able to match it to a more precise location than the 5-digit ZIP centroid.

Preferring the User Dictionary

When the preferences are set to Preferring the User Dictionary, candidates from the User Dictionary are given a higher score than a similar candidate from the Address Dictionary. This setting is designed for use only in instances where the user feels their User Dictionary is much superior to the MapMarker Address Dictionary.

Viewing Match Candidates on a Map

Candidate visualization allows you to see where potential matches fall on a map to help you determine the best match for your address. This feature is accessible via a MAP button in the Interactive Geocode and Quick Find dialog boxes.

Use candidate visualization during interactive geocoding to select the point on the map that represents your match choice. MapMarker geocodes to that record. The Quick Find map feature allows you to view the candidates on a map as a quick way to confirm an address.

Setting the Street Data Path

Candidate visualization uses MapMarker Streets, StreetInfo, or StreetPro tables as the underlying street network in the Map window. Prior to mapping your candidates, specify the location of your street data in the Map tab of the SYSTEM PREFERENCES dialog box. You may also control which layers you want displayed in the map.

Viewing Match Candidates During Interactive Geocoding

To view potential matches on a map during interactive geocoding:

1. From the INTERACTIVE dialog box, proceed through the table until you reach a record with a list of match candidates that you want to view on a map.
2. Click the MAP button. MapMarker creates a map view of all match candidates in the list. Each candidate is displayed with the color that represents its type of match. For example, green stars are candidates that match to the street address (S5) and yellow stars are those that match to the ZIP+4 centroid (S3 or Z3).
3. Use the tools provided to pan, zoom, and label the map to help you decide the best match candidate for the address. The label tool allows you to label nearby streets and features to provide a geographic reference for the candidates.
4. When you decide the best match, click on the candidate symbol with the Select tool and click OK. You are returned to the INTERACTIVE dialog box. Your choice is highlighted in the candidate list.

5. Click GEOCODE to geocode your address to the candidate.

**Viewing Match Candidates Via Quick Find**

To view potential matches on a map from Quick Find:

1. From the SEARCH>QUICK FIND dialog box type in an address you wish to find in the Address Dictionary. Click SEARCH.

2. If MapMarker returns one or more match candidates, click the MAP button to view them on a map. The Candidate Map window is displayed.

3. Use the pan, zoom, and label tools to help you identify the best match for your address. Use the label tool to identify surrounding streets and features for additional information about the area.

4. With the Select tool, click on the candidate symbol for your match choice and click OK. You are returned to the QUICK FIND dialog box. The candidate is highlighted in the list. Make a note of the corrected address and click CLOSE. Note that Quick Find is only a tool to verify a match. It does not geocode.

**Considerations**

MapMarker must return a list of match candidates in order to create a Candidate Map view to be created.

When mapping candidates from the INTERACTIVE dialog box, you can use all the features of the dialog box to maximize the potential for a match. For example, you may wish to expand the search area so that MapMarker returns more match candidates for viewing. In the QUICK FIND dialog box, you are limited to editing the address to try for additional or different match candidates.

You cannot map candidates when you click BROWSE from the QUICK FIND dialog box.

**Point Synchronicity with Street Data: MapMarker vs. MapMarker Plus**

For proper placement of the match candidates in relation to a street network, be mindful of the geocoding and Street products you use.

If you geocode your table with the standard MapMarker product, be sure to display those records over MapMarker Streets, StreetInfo, or StreetPro Display street products.

If you geocode your table with MapMarker Plus, you should display the records over MapMarker Plus Streets or StreetPro Enhanced products.
A Closer Look at Multiple Matches

A multiple match is when MapMarker finds more than one record in the Address Dictionary that cover the same address range. You have complete control over how you want MapMarker to respond when it encounters a multiple match. In the Multiple Match tab of the Geocode dialog box the following choices are available.

If you choose the default setting, **DO NOT MATCH**, MapMarker does not geocode the record if it finds a multiple match. During interactive geocoding, MapMarker stops at that record and waits for your response. Sometimes it finds multiple matches, even though there is a candidate that appears to be an exact match. In this situation it is important to examine the ranges associated with the candidate in order to make the best judgement about whether to match to the candidate.

Consider the following example.

MapMarker determined that there are multiple matches for 981 Howard St. in San Francisco, even though it appears one candidate matches the record. By clicking the **RANGES** button and scrolling through the list of ranges, you see that there are four address ranges shown that cover 981 Howard St.

Because the multiple match setting is set to Do Not Match, MapMarker stops geocoding at this point and waits for you to choose the match or ignore it. By clicking on the **GEOCODE** button in this example, MapMarker generates an S result code, which means a single close match was made.

If the restriction had been set to one of the other choices in the **MULTIPLE MATCH** tab, this record would not be displayed in the **INTERACTIVE** dialog box because MapMarker would geocode it automatically. If the setting was Accept First, Pick Street Address over ZIP+4, or Pick ZIP+4 over Street Address, the result code would be an M result. For more on result codes, see **Understanding Result Codes on page 62**.
Result Codes

In this chapter:

- Understanding Result Codes ......................................................... 62
- Locational Order of Accuracy ......................................................... 66
- S3 and Z3 Result Codes: What’s the Difference? .............................. 67
- Interpreting S5 Result Codes ............................................................ 67
Understanding Result Codes

As an output option, MapMarker returns a result code for every record it attempts to match. The code indicates the success or failure of the geocoding operation as well as conveys information about the quality of the match. Each character of the code indicates the level of precision of each address component matched.

The result codes are written to a result column specified in the SELECT OUTPUT COLUMNS dialog box at the beginning of the geocoding process. You can either create this column before geocoding or have MapMarker create it automatically (default name GeoResult) when you open your table.

The code is an alphanumeric code of 1–10 characters. The codes fall into four major categories:

- Single unique match
- Best match from multiple candidates
- ZIP Code centroid match
- Non-match

The codes reflect the settings you choose or accept in the GEOCODE dialog box. They also reflect the quality of the data in both your table to be geocoded and the Address Dictionary. Each category is explained below.
**Single Match (S category)**

Matches in the S category indicate that the record was matched to a single address candidate. The first character (S) reflects that MapMarker found a street address that matches your record.

When 5-digit ZIP Codes are without an area, they are dots on a map rather than polygons, and have no geographic extent defined in terms of street segments. These point ZIPs include P.O. Box ZIPs and Unique ZIPs (single site, building, or organization). Instead of returning an S1 result code from geocoding, S6 has been introduced to MapMarker for these special ZIPs, as they represent actual locations.

The second position in the code reflects the positional accuracy of the resulting point for the geocoded record, as indicated below.

- **S1** – single match, point located at ZIP Code centroid
- **S2** – single match, point located at ZIP+2 centroid
- **S3** – single match, point located at ZIP+4 centroid
- **S4** – single match, point located at the center of a shape point path (shape points define the shape of the street polyline)
- **S5** – single match, point located at a street address position (highest accuracy available)
- **S6** – single match, point located at point ZIP centroid
- **SX** – single match, point located at street intersection
- **S0** – single match, no coordinates available (very rare occurrence).

When you use the MapMarker desktop to geocode, you will either get a close match, or a non-match. When using the APIs it is possible for non-close matches to be returned.
Best Match From Multiple Candidates (M category)

Matches in the M category indicate that there is more than one match candidate for the record and MapMarker has chosen the best one of those candidates. This category is used when you select Accept First, Pick Street Address over ZIP+4, or Pick ZIP+4 over Street Address in the MULTIPLE MATCH tab and MapMarker finds more than one strong match candidate.

As in the S category, the second position in the code of M category matches the positional accuracy of the resulting point object.

- **M1** – multiple matches, point located at ZIP Code centroid
- **M2** – multiple matches, point located at ZIP+2 centroid
- **M3** – multiple matches, point located at ZIP+4 centroid
- **M4** – multiple matches, point located at the center of a shape point path (shape points define the shape of the street polyline)
- **M5** – multiple matches, point located at a street address position (highest accuracy available)
- **M6** – multiple matches, point located at point zip location
- **MX** – multiple matches, point located at street intersection
- **M0** – multiple matches, no coordinates available

For either S or M category result codes, eight additional characters describe how closely the address in your table matches an address in the Address Dictionary. The characters appear in the order given. Any non-matched components are represented by a dash.

<table>
<thead>
<tr>
<th>Result Code Component</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>House Number</td>
<td>110</td>
</tr>
<tr>
<td>P</td>
<td>Street Prefix</td>
<td>North</td>
</tr>
<tr>
<td>N</td>
<td>Street Name</td>
<td>Fletcher</td>
</tr>
<tr>
<td>T</td>
<td>Street Type</td>
<td>Place</td>
</tr>
<tr>
<td>S</td>
<td>Street Suffix</td>
<td>SE</td>
</tr>
<tr>
<td>C</td>
<td>City Name</td>
<td>Boulder</td>
</tr>
<tr>
<td>Z</td>
<td>ZIP Code</td>
<td>80303</td>
</tr>
<tr>
<td>A or U</td>
<td>Address Dictionary or User Dictionary</td>
<td>A</td>
</tr>
</tbody>
</table>

For example, the result code S5- -N-SCZA represents a single close match that matched the street name, street suffix direction, city and ZIP Code exactly, but could not match the house number, street prefix direction, or the street type. The match came from the MapMarker Address Dictionary. This record would spot at the street address position of the match candidate.
Postal Centroid Matches (Z category)

Matches in the Z category indicate that no street match was made, either: 1) because there is no close match and you allowed MapMarker to fall back to ZIP Centroid; 2) the address is a P.O. Box or rural address; or 3) you set MapMarker to match to ZIP Centroids. The resulting point is located at the ZIP Code centroid with five possible accuracy levels.

- **Z1** – ZIP Code centroid match
- **Z2** – ZIP+2 centroid match
- **Z3** – ZIP+4 centroid match (highest accuracy available)
- **Z0** – ZIP Code match, no coordinates available (very rare)
- **Z6** – ZIP Code centroid match for point ZIP

When 5-digit ZIP Codes are without an area, they are dots on a map rather than polygons, and have no geographic extent defined in terms of street segments. These point ZIPs include P.O. Box ZIPs and Unique ZIPs (single site, building, or organization). Instead of returning a Z1 result code from geocoding, Z6 has been introduced to MapMarker for these special ZIPs, as they represent actual locations.

Non-match Codes

The following result codes indicate no match was made:

- **N** – No close match. These records can be re-geocoded interactively or during subsequent automatic passes under different matching conditions.
- **NX** – No close match for street intersections.
- **ND** – MapMarker could not find the Address Dictionary for the given ZIP Code or city/state. These records can be re-geocoded once the Address Dictionary is available. An ND code can also appear in the fields of ZIP+4 output columns, e.g., MapMarker returns 12180-23ND. In this case the ND code represents ZIP+4 Codes associated with nondelivery areas such as vacant lots or other areas to which the US Postal Service does not deliver mail.
- **NG** – The user marks these records during interactive geocoding as non-geocodable. MapMarker does not attempt to match these records again until the code is removed.

GeoResult Dialog Box

To view a breakdown of the result code for each record, double-click on the code in the result column in the Browser. This behavior is available after you have completed a geocoding pass or set the columns for a previously geocoded table. It is not available at the initial opening of the table.
Information about the record shows the address, result code, match type and match details by address component. The dialog box is modeless; that is, you can continue to click on result codes in your table while this dialog box is open and the result code information for that record is displayed.

**Note:** You may notice that in the above illustration, the street prefix and suffix directions are selected, indicating the record matched on those components. The address 280 NW 41st St. only contains a prefix direction. MapMarker considers it a match when both the record to geocode and the match candidate from the Address Dictionary have the same components. In this case, although neither address contain a suffix, MapMarker considers it a match on that component.

To increase the number of successful matches, using MapInfo, search your table for records containing the same code. You can then edit and re-geocode them to increase the likelihood of a match.

### Locational Order of Accuracy

The MapMarker Plus USA engine scores results from highest level of accuracy to lowest, in this order:

- S6
- S5
- S4
- S3
- S2
- S1
- S0
S3 and Z3 Result Codes: What’s the Difference?

One of the most often asked questions about MapMarker results is what is the difference between S3 and Z3 result codes.

An S3 match is defined as a single close match with the point located at the ZIP+4 centroid. The Z3 match is also spotted at the ZIP+4 centroid. The difference lies in how MapMarker arrives at the results.

For the S3 record, MapMarker found a street address that matches, but the match record did not contain any street geometry (line points). MapMarker is, therefore, unable to interpolate where along the segment to place the record. The best it can do is to spot it at the ZIP+4 centroid.

A Z3 match, on the other hand, is a direct match to the ZIP+4 centroid. MapMarker could not find a street address match for one of several reasons: 1) you set the Geocode Precision to To ZIP Centroid; 2) there was no close match and you set your fallback criteria to ZIP Centroid; or 3) the address is a P.O. Box or rural route.

On a map, the S3 and Z3 records display at the same location, assuming the input ZIP Codes were the same for both records. However, the matching process for an S3 record has an extra step that could place the point at a different, and more accurate, ZIP+4 centroid. During a street address match, MapMarker corrects the input ZIP Code if the match record includes different information. Z3 matches do not get corrected. Because of this correction step, the S3 match is considered the more precise match.

Interpreting S5 Result Codes

If you receive an S5 match for a record that did not spot where you thought it should, you need to analyze the result code closely to determine why. This situation occurs during a geocoding pass where the match conditions are relaxed, that is, you did not require an exact match on house number, street name, and/or ZIP Code. When MapMarker attempts to geocode the record with those settings, it searches larger areas than it would if the conditions were stricter. It may match the record to a distant address that has a similar name.

To understand how MapMarker made the match, look at the full result code. A typical result in this situation looks like this: S5-P--S--A. Even though MapMarker returned an S5 (it found a single match to a street address), it did not match the house number (H), street name (N), street type (T), city (C), or ZIP Code (Z). The missing components in the result code are represented by dashes. A perfect street level match would return a result code with every component of the address matched and would look like S5HPNTSCZA.

In order to minimize the matches that spot far afield of where they should be, geocode your table with stricter matching conditions. For the strictest match settings and greatest matching accuracy, try geocoding in CASS mode. CASS geocoding is explained in Geocoding for CASS Certification on page 70.
This chapter looks at a number of specialized features of MapMarker, including geocoding for CASS certification, interactive geocoding, Quick Geocoding, Quick Find, Locate Highway Exit, and Locate Airport. Instructions for batch geocoding and choosing the right datum are also included.

In this chapter:

- Geocoding for CASS Certification ................................................. 70
- Delivery Point Validation (DPV) ..................................................... 71
- Geocoding to ZIP Code Centroids ............................................... 77
- Geocoding dBASE Files ................................................................. 78
- Using Quick Geocode ................................................................. 78
- Finding a Single Address with Quick Find .................................... 78
- Finding Airports with Locate Airport .......................................... 81
- Finding Highway Exits with Locate Highway Exit ....................... 82
- Batch Geocoding ........................................................................... 83
- Table Attribution with MapMarker ............................................. 85
Geocoding for CASS Certification

CASS certification is a process by which a table of mailing addresses is standardized to meet U.S. Postal Service requirements for bulk mailing discounts. MapMarker Plus performs this address standardization under strict matching conditions set by the USPS while it geocodes your records.

You may geocode using the CASS standards for any table, local or remote, as long as the proper output columns are provided. You must have the current revision of the MapMarker Address Dictionary available in order to comply with CASS address standardization. Look in the About Box under the Help menu for the Data Revision date.

Note: You cannot geocode your records to a user dictionary during CASS geocoding.

To geocode your data for CASS certification:

1. Add the following output columns to your source table. You can name the columns to any name you like. Make sure the type and length are as follows:

   **Required Fields**
   - Firm – Character(30)
   - Address – Character(64)
   - Address2 – Character(64)
   - City – Character(30)
   - State – Character(2)
   - Zip – Character(9)
   - Zip+4 – Character(4)
   - URB – Character(30) [This field is used to store urbanization codes for Puerto Rican addresses. You must have this field even if you do not have Puerto Rican addresses.]
   - Delivery Point – Character(2)
   - Check Digit – Character(1)
   - LACS – Character(1)
   - PMB – Character(4)
   - PMB Range – Character(8)

   **Optional Fields**
   - Latitude – Float OR Decimal(11,6)
   - Longitude – Float OR Decimal(11,6)
   - Census – Character(15)
   - Carrier Route – Character(4)
   - DefaultFlag – Character(1)
   - EWS Flag – Character(1)
   - RecordType – Character(1)

2. Start MapMarker.

3. Open the table you want to geocode. Proceed through the **SELECT COLUMN** dialog boxes.

4. On the **TABLE** menu, click **GEOCODE** and then click the **CASS** tab.
5. Select **OVERRIDE EXISTING GEOCODING SETTINGS WITH CASS DEFAULTS**. MapMarker uses these defaults to geocode your data to CASS specifications.

6. Use of the **USE EWS FILE** check box is optional. Any address found in the EWS file is automatically marked as unmatched.

7. To produce a US Postal Service Form 3553 CASS Report select the **PRODUCE A CASS REPORT** check box. You will be prompted for a name and location for the report at the **SAVE CASS REPORT AS** dialog box.

8. When you are returned to the **CASS** tab, click **GEOCODE** to begin the operation.

MapMarker geocodes your data to CASS standards and provides a printed report of the address standardization operation (optional).

**Delivery Point Validation (DPV)**

The Delivery Point Validation (DPV) option that you may have purchased with MapMarker is technology available from the U.S. Postal Service (USPS). Used in conjunction with CASS mode geocoding, DPV enables you to verify whether an address is a deliverable USPS address.

Using DPV will enable you to determine whether an address exists before your mailing leaves the premises. Conversely, you will also be able to identify those addresses that are potentially undeliverable as addressed because of faulty address information.

This section explains how to unlock DPV, how to use DPV when you geocode in CASS mode, and its usage requirements and restrictions.

**Unlocking DPV**

Customers who purchase DPV must obtain an unlocking code from MapInfo’s Customer Service department before they can use the feature. To obtain the unlocking code, customers must generate a system key on the computer where MapMarker is installed and provide the key to MapInfo Customer Service.
The DPV Unlocker application that comes with MapMarker generates the key and unlocks the DPV data once the unlocking code is entered into the application. You will need to run the DPV Unlocker before you use DPV for the first time. You may also need to run the DPV Unlocker to re-enable DPV after a usage violation. See Usage Requirements on page 76 for more specific information.

Unlocking Code

The DPV unlocking code contains the following encrypted information:

- Unlocking code status
- DPV expiration date
- Unlocking code expiration date

The unlocking code status can be one of the following values:

- Create—The unlocking code is valid for a first-time unlocking of DPV.
- Repair—The unlocking code is valid for a first-time re-enable of DPV.

The DPV expiration date is the date that the DPV data expires. The unlocking code is valid for 14 days.

Unlocking Requirements

All of the following conditions must be met before you can unlock DPV:

- Unlocking code is not expired (unlocking codes are valid for 14 days).
- The DPV expiration date has not passed.
- The DPV expiration date matches the CASS expiration date.
- The unlocking code status is appropriate for the DPV status. There are two acceptable combinations of DPV and unlocking code status:
  - First-time unlocking scenario (DPV status is “None” and the unlocking code status is “Create”).
  - First-time re-enable scenario (DPV status is “Disabled” and the unlocking code status is “Repair”).

Unlocking DPV for the First Time

During the MapMarker installation, the DPV Unlocker application is placed into the MapMarker directory. You can launch it directly from the Windows Start menu. To obtain the DPV unlocking code:

1. Open the DPV Unlocker application via the Windows Start menu. It is located in your MapMarker directory. If you accepted all the default locations during the installation, you would choose Start>Program Files>MapInfo>MapMarker>DPV Unlocker.
2. In the DPV Unlocker application window, you will see a System Key displayed in the System Key box. Call MapInfo Customer Service at 1-800-552-2511 and supply the Customer Service Representative with the key. They will in turn provide you with an unlocking code.

3. Enter the unlocking code into the Unlocking Code box.

4. Click Unlock DPV.

5. Click Close to close the program.

Reactivating DPV

If you incur a usage violation while geocoding with DPV mode, DPV is automatically shut off. As part of your DPV license, you are entitled to a one-time re-enable of the feature. This requires that you obtain a new unlocking from Customer Service. Use the instructions above to run the DPV Unlocker application.

When you generate the system key, it will be longer than your original system key because it holds information about the violation that prompted the shutoff.

Subsequent DPV usage violation require that you contact the U.S. Postal Service. For more information, see Data Security and Improper Use on page 76.

Using DPV

DPV is used as part of CASS to validate that geocoded addresses are deliverable USPS addresses. Geocoding with DPV is available only when you are running in CASS mode, and requires extra columns in your table to hold the DPV output. This section explains the DPV columns and how to set up your table to geocode in DPV mode.

DPV Columns

DPV generates information for each geocoded address during the geocoding process. To hold this information, you must add extra columns to your table before using the DPV feature.

There are nine columns you must add. These columns contain information specific to DPV, as well as a number of USPS footnote codes that give more information about how the address was validated.

You can add these columns to your table beforehand, or when you are ready to select columns, using the Modify/Add Columns button in the Select output columns from your table dialog. Clicking this button opens the Modify Table Structure dialog, where you can add the necessary columns.
To select the DPV columns, click the Show DPV Columns button in the Select output columns from your table dialog. The DPV columns display in the dialog.

```
Select output columns from your table
```

Selecting DPV columns uses the same process as selecting other output columns to prepare for your geocoding session. Click and drag the column name to the appropriate box to select the column. To return to the main part of the dialog, click Hide DPV Columns. The button toggles back and forth between Show and Hide.

The DPV columns are described below:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfCode</td>
<td>The DPV confirmation code for the geocoded address. The confirmation code can be one of the following: N–This address does not exist. Y–This address exists. S–The street address exists, but the unit does not. D–Address is incomplete (highrise with no unit, or rural route with no box number).</td>
</tr>
<tr>
<td>Cmra</td>
<td>Indicates if the geocoded address belongs to a Commercial Mail Receiving Agency (CMRA)</td>
</tr>
<tr>
<td>falsePos</td>
<td>If true, DPV has detected a condition in which the geocoded address appears to be artificially generated and not a legitimately obtained address. DPV will shut down immediately in this instance.</td>
</tr>
<tr>
<td>footnote1</td>
<td>Field that may be populated with a standard USPS footnote code. Each of the codes is described below.</td>
</tr>
</tbody>
</table>
Footnote Codes

The following DPV footnote codes may be output for match candidates:

- AA–Input address matched to the ZIP+4 file.
- A1–Input address not matched to the ZIP+4 file.
- BB–Input address matched to DPV (all components).
- CC–Input address primary number matched to DPV but secondary number not matched (present but invalid).
- N1–Input address primary number matched to DPV but highrise address missing secondary number.
- M1–Input address primary number missing.
- M3–Input address primary number invalid.
- P1–Input address missing PO, RR, or HC box number.
- RR–Input address matched to CMRA.
- R1–Input address matched to CMRA but secondary number not present.

DPV Mode

When you geocode in DPV mode, MapMarker checks each geocoded address against the DPV database to verify whether the address exists. The DPV database consists of over 145 million USPS-delivered addresses, and is made available by the USPS for this specific use as part of the DPV option you purchased.

To activate DPV, you must first be running in CASS mode. In the CASS tab of the Geocode dialog, select the Override Existing Geocode Settings with CASS Defaults check box to turn on CASS mode. When you do this, all the subsequent check boxes become available. Next, select the Use DPV mode check box. You are now ready to geocode using DPV.
Usage Requirements

The DPV feature has specific requirements imposed by the USPS CASS certification process to ensure the validity of the data, its security, and to guard against improper use of the feature. This section explains the mechanisms MapMarker has in place to meet these requirements.

DPV Data Expiration

MapMarker ships quarterly to comply with CASS requirements. We will ship the most current DPV data available, but keep in mind that the CASS data and the DPV data are on different update schedules: CASS is quarterly; DPV is bimonthly.

Because the operation of DPV is so closely tied to the currentness of the CASS data, we continue to ship quarterly align ourselves with CASS. We will make every effort to provide you with the most up to date DPV data available at the time that MapMarker ships.

Data Security and Improper Use

To protect the security of the DPV data, the U.S. Postal Service requires that we develop a stop processing sequence that will shut down DPV if it detects a possible misuse of the DPV data. It is designed to guard against users who are attempting to create address lists with the data, rather than perform true address validation. The USPS has asked that we publish the following information regarding the proper use of the feature, in the event that DPV is shut off:

DPV processing was terminated due to the detection of what is determined to be an artificially created address. No address beyond this point has been DPV validated. In accordance with the License Agreement between USPS and MapInfo Corporation, DPV shall be used to validate legitimately obtained addresses only, and shall not be used for the purpose of artificially creating address lists. The written Agreement between MapInfo Corporation and its customers shall also include this same restriction against using DPV to artificially create address lists. Continuing use of DPV requires compliance with all terms of the License Agreement. If you believe this address was identified in error, please contact MapInfo Corporation.
Stop Processing Sequence

The stop processing sequence is triggered if DPV encounters a match in the false positives table in the address validation process. This table, supplied by the USPS, contains a list of false positive addresses. Whenever MapMarker cannot find an address in the main DPV table, it checks the address against the false positives table. If the address is in the false positives table, DPV processing stops, although you will still be able to continue with your CASS geocoding session. MapMarker saves the address record that triggered the false positive and generates a log so that you can identify the address.

First Instance of DPV Stop Processing

The first time MapMarker encounters a false positive address and DPV is shut off, the address must be sent to the USPS.

You are entitled to a one-time restart of DPV the first time an address matches an address in the false positives table. To reactivate DPV, you must run the DPV Unlocker application again to generate another system key and call MapInfo Customer Service to obtain the unlocking code. MapInfo must then report your company name and address to the USPS.

Second Instance of DPV Stop Processing

If DPV is shut off a second time, you must perform the same procedure of running the DPV Unlocker and calling MapInfo Customer Service to provide the Customer Service representative with the system key. However, on a second shut down, MapInfo must contact the USPS and obtain permission to reactivate the license before you can receive another unlocking code.

Geocoding to ZIP Code Centroids

As an alternative to street-level geocoding, you can specify for MapMarker to geocode your records to the appropriate ZIP Code centroid. This is a fast method of geocoding because it eliminates the need to match based on street address. This method matches the ZIP Code in your table with the ZIP Code in the Address Dictionary. MapMarker geocodes to the most precise ZIP Code level contained in your table.

Use ZIP Code centroid matching when you have a large database that you want to geocode quickly and you are less concerned with spotting the points to street level.

To geocode to ZIP Code centroids:

1. In the Geocode dialog box, choose TO ZIP CENTROID under GEOCODE PRECISION.
2. Choose the range and type of rows you want to geocode.
3. Click GEOCODE. MapMarker updates the AUTOMATIC GEOCODE dialog box every 25 records.
4. When through, click DONE to view the Log file or return to the MapMarker menu.
Geocoding dBASE Files

MapMarker reads dBASE format (.dbf) files as well as MapInfo native format (.tab), thus enabling you to turn any table of records into geographic objects. This section explains some things to keep in mind when geocoding *.dbf files.

MapMarker creates a *.tab file automatically when you open a *.dbf file. If you open the *.dbf a second time, a dialog box is displayed stating that a *.tab file already exists. Choose Yes to replace it.

Since MapMarker automatically creates a *.tab file for any *.dbf file you open, all of the options available to *.tab files apply to *.dbf files. Among them, MapMarker adds coordinate and result columns and creates points for your table automatically.

MapMarker does not display memo fields included with *.dbf files. The fields are not changed in any way by MapMarker.

MapMarker does not maintain indexes for *.dbf files. If you write to an indexed field during geocoding, the table must be re-indexed outside of MapMarker if you wish to maintain the index.

Using Quick Geocode

Quick Geocode is a shortcut to geocoding your table with MapMarker. Use it to geocode previously opened tables with a single click. For example, use Quick Geocode on tables that you update frequently and need to geocode new records. Quick Geocode follows the geocoding preferences and system preferences settings you specify to carry out the geocoding operation.

Quick Geocode is available after you open a table in MapMarker and set the input and output columns, geocoding, and system preferences. You can geocode the entire table or a portion of the table. When you close the table, the settings are written to the table’s metadata and the system registry, preserving your preferences for the next time you need to geocode the table.

Note: MapMarker attempts to geocode all records in the table during a Quick Geocode pass, regardless of their previous geocoding status.

Finding a Single Address with Quick Find

MapMarker’s Quick Find feature is a fast way to determine the correct address for a single record. This feature does not geocode the record; it simply attempts to match the address to the Address Dictionary and return potential matches that you can use to confirm or correct your address. If there are several match candidates to choose from, view them on a map to help decide the best choice. Quick Find is under the Search menu and is available whenever MapMarker is running.
Quick Find can also be used as a method to browse the Address Dictionary for the correct spelling of a street name or firm. Provide the ZIP Code and at least the first letter of the street or firm and click the **BROWSE** button. The city and state fields are optional. MapMarker returns a list of streets or firms that meet your criteria.

To maximize MapMarker’s ability to return a list of browse candidates, do not provide a complete street address with house number. This Browse feature only returns street or firm names. Use the Search capability in Quick Find to find matches to street names with house numbers. The **MAP** and **RANGES** buttons are unavailable during a Browse.

**Quick Find Settings**

You can change the criteria used in producing your Quick Find search results in the Quick Find Settings dialog box. You can select which street address input values to use in finding an exact match, choose whether to list all candidates or close candidates, and whether or not to match on intersections, use CASS mode, or to relax internal matching rules.
To change the Quick Find settings:

1. Open the Quick Find dialog box (see Finding a Single Address with Quick Find on page 78).
2. Click QUICK FIND SETTINGS.
   The Quick Find Settings dialog box displays.
3. Select the settings you wish to use and then click OK.

<table>
<thead>
<tr>
<th>Quick Find Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>House Number</td>
<td>Click this check box to require the house number be used in determining an exact match.</td>
</tr>
<tr>
<td>Street Name</td>
<td>Click this check box to require the street name be used in determining an exact match.</td>
</tr>
<tr>
<td>City Name</td>
<td>Click this check box to require the city name be used in determining an exact match.</td>
</tr>
<tr>
<td>ZIP Code</td>
<td>Click this check box to require the ZIP code be used in determining an exact match.</td>
</tr>
<tr>
<td>Match Intersections</td>
<td>Click this check box to allow matching on intersections.</td>
</tr>
<tr>
<td>Use CASS Mode</td>
<td>Click this check box to use CASS mode in finding addresses. <strong>Note:</strong> When this check box is selected, the other settings options on this dialog are disabled and their values are overwritten by CASS rules. List Candidates options (All Candidates and Close Candidates Only) are still available for selection.</td>
</tr>
<tr>
<td>Relax Internal Matching Rules</td>
<td>Click this check box to relax internal matching rules used by default, allowing for a larger pool of addresses to be geocoded using the preference settings you define.</td>
</tr>
<tr>
<td>All Candidates</td>
<td>Click this option to return all candidates matching the search criteria you specify.</td>
</tr>
<tr>
<td>Close Candidates Only</td>
<td>Click this option to return only the candidates that closely match the search criteria you specify.</td>
</tr>
</tbody>
</table>
MapMarker’s Locate Airport feature allows you to locate an airport using a state abbreviation or a specific airport code. Enter an airport code or two-letter state abbreviation in the text field of the Search frame. The Search Results box returns the number of successful matches for a state, match found for an airport code, or no matching records found. After a successful match, the Airport List field displays the airport code, airport name, state, and latitude/longitude coordinates.

In the event a match cannot be made, the Search Results box displays one of three messages depending on your search criteria:

<table>
<thead>
<tr>
<th>Search Results Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid state or location code.</td>
<td>The airport or state code you searched on did not contain at least two alpha-numeric characters. Re-type the airport or state code, using two characters if searching by state, or three to four characters if searching by airport code.</td>
</tr>
<tr>
<td>Invalid state code.</td>
<td>The two-character code you searched on did not match a valid state code. Type the correct state code to search for all airports in the state or type the complete three- or four-character airport code to search for a specific airport.</td>
</tr>
<tr>
<td>No matching records found.</td>
<td>The three- or four-character code you searched on did not match a valid airport code. Type the correct airport code to search for the airport or type a state code to search for all airports in the indicated state.</td>
</tr>
</tbody>
</table>

To locate an airport:

1. On the SEARCH menu, click LOCATE AIRPORT. The Locate Airport dialog box displays.
2. Type the airport code or the state abbreviation the AIRPORT CODE OR STATE ABBREVIATION field (for example, ALB for Albany International Airport or NY for all airports in New York).
3. Click **SEARCH**.

MapMarker returns all matching airports in the Airport List box, sorted by airport code.

---

**Finding Highway Exits with Locate Highway Exit**

MapMarker’s Locate Highway Exit feature allows you to locate a highway exit using the highway name, exit number, and state. You must enter a highway name, an exit number or exit name, and a state in order for the search to return a result. Search results include the highway name, full exit name, city, state, zip code, and X and Y coordinates for the exit. You can select a search result and click the Map button to view the candidate’s location on a map to determine which candidate is the exact one you are looking for.

To locate a highway exit:

1. On the **SEARCH** menu, click **LOCATE HIGHWAY EXIT**. The Locate Highway Exit dialog box displays.
2. Type a highway name in the **HIGHWAY** field (for example, **I-90 East** or **Interstate 90 East**).
3. Type an exit number in the **EXIT** field (for example **2** or **Exit 2 to Washington Ave.**).
4. Type a state abbreviation in the **STATE** field (for example, **NY** for New York).
5. Click **SEARCH**.

MapMarker returns all matching highway exits in the Search Results box.
Batch Geocoding

MapMarker supports batch geocoding for processing one or more tables without constant user interaction. You set up a text file containing the path and file names of the tables to geocode and specify the geocoding preferences you want applied to the session.

Batch geocoding is useful for overnight processing of large databases or when a table is updated regularly via a scheduling program.

Create the batch file from the MapMarker interface by opening a table and selecting the geocoding preferences.

To create a batch file:

1. Open a table in MapMarker and proceed through the select column dialog boxes to confirm the input and output columns.
2. If you wish to add attributes to your table when you are geocoding it, point to **ADD ATTRIBUTES** on the **TABLE** menu and click **SELECT ATTRIBUTE TABLES** to set up the attribution process. See Table Attribution with MapMarker on page 85 for detailed instructions on adding attributes.
3. In the **GEOCODE** dialog box, choose the settings you want for the geocoding pass. Choose from the Geocode, Log File, Fallback, and Multiple Match tabs. You cannot choose CASS settings for batch geocoding. After selecting the restrictions, click **APPLY**, then click **CANCEL**. Click the **HELP** button in the dialog box to learn more about each geocoding option.
4. When you are finished with the settings, click **CREATE BATCH FILE** on the **TABLE** menu. The Save Batch File As dialog box is displayed.

5. Specify the path and filename for the batch file. By default the batch file uses the root file name of the table with the extension *.bat*. Click **SAVE**.
6. Repeat steps 1–5 for each additional table you want included in the batch file. When saving it to the batch file, specify the same name you created in step 5. To create a new batch file, use a new filename.
7. To run the batch file, double-click on the filename in Explorer. MapMarker starts running and opens each table listed in the batch file in turn and then geocodes it to the specified settings. If MapMarker encounters an error, it stops.
Creating a Batch File By Hand

You can also create the batch file by hand, if you prefer. In a text editor, list the path and filename for the table and any geocoding parameters you want MapMarker to follow. The list of parameters is shown below. In order to activate the parameters, open the table in MapMarker at least once and set the preferences. Otherwise MapMarker geocodes to the default settings for that table as found in the metadata.

The command line parameters are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/TABLE=c:\sample\table</td>
<td>Path and filename of table to geocode.</td>
</tr>
<tr>
<td>/LOG=&lt;logfile name&gt;</td>
<td>Name of log file. MapMarker writes the geocoding results to this file. If a log file is not specified, MapMarker creates a default Mapmarkr.log file.</td>
</tr>
<tr>
<td>/STREET or /ZIP</td>
<td>Specify geocode precision to street level or to ZIP Code centroids.</td>
</tr>
<tr>
<td>/START=&lt;number&gt;</td>
<td>Specify the starting row number to geocode. If omitted, MapMarker begins with row 1.</td>
</tr>
<tr>
<td>/END=&lt;number&gt;</td>
<td>Specify the ending row number to geocode. If omitted, MapMarker ends at the end of the table.</td>
</tr>
<tr>
<td>/ALL or /UNMATCHED</td>
<td>Specify whether to geocode all records or only unmatched records.</td>
</tr>
<tr>
<td>/EXACT_HOUSE=Y,N</td>
<td>Specify yes or no to geocode to the exact house number.</td>
</tr>
<tr>
<td>/EXACT_STREET=Y,N</td>
<td>Specify yes or no to geocode to the exact street name.</td>
</tr>
<tr>
<td>/EXACT_CITY=Y,N</td>
<td>Specify yes or no to geocode to the exact city name.</td>
</tr>
<tr>
<td>/EXACT_ZIP=Y,N</td>
<td>Specify yes or no to geocode to the exact ZIP Code.</td>
</tr>
<tr>
<td>/XSECT=Y,N</td>
<td>Specify yes or no to geocode to street intersections.</td>
</tr>
<tr>
<td>/STREETONLY=Y,N</td>
<td>Specify yes or no to geocode only to street addresses, not to ZIP Codes.</td>
</tr>
<tr>
<td>/STREET_PTZIPONLY=Y,N</td>
<td>Specify yes or no to geocode to exact street address and Point Zip.</td>
</tr>
<tr>
<td>/FALLBACK=NO,ZIP4,ZIP2,ZIP</td>
<td>Specify whether MapMarker falls back to a ZIP Code, and if so, to which precision: ZIP+4, ZIP+2, or ZIP Code.</td>
</tr>
<tr>
<td>/MULTI=NO,FIRST,STREET,ZIP</td>
<td>Specify how MapMarker handles multiple matches:</td>
</tr>
<tr>
<td></td>
<td>• No = do not geocode</td>
</tr>
<tr>
<td></td>
<td>• First = geocode to the first multiple match</td>
</tr>
<tr>
<td></td>
<td>• Street = choose a street match over a ZIP Code match</td>
</tr>
<tr>
<td></td>
<td>• ZIP = choose a ZIP Code match over a street match.</td>
</tr>
</tbody>
</table>
Batch Geocoding Considerations

If you list a table in the batch file without parameters, MapMarker geocodes to the default settings for the table (whatever the settings were the last time the table was opened in MapMarker).

MapMarker's batch geocoder uses the system preferences that are set at the time of geocoding. To ensure that a particular system setting is used, such as the location of the Address Dictionary, choose the setting in the **SYSTEM PREFERENCES** dialog box, apply it, make the batch file, and run the batch file during the same MapMarker session.

MapMarker uses the input and output columns it finds in the table's metadata. Columns cannot be changed for batch geocoding. CASS geocoding is not supported in MapMarker's batch geocoder.

For a hand-generated batch file, you can use uppercase, lowercase, or mixed case text. The parameters can be listed in any order.

Table Attribution with MapMarker

MapMarker has the ability to add information from one table to another, either before, during, or after the geocoding process. This ability is called attribution.

Any table in MapInfo format can be a candidate for attribution: boundary and point files, demographic tables, or non-geographic tables. The table to which the attributes are being applied must be local, as it is the table you will be geocoding. The table containing the source attributes can be local or remote.

There are two ways to perform attribution in MapMarker. They are defined by the type of link required to connect the two tables:

- **Output Point attribution** – links the geocoded table to the attribution table by geography, specifically by matching latitude and longitude point coordinates within the geocoded table to a geographic boundary defined within the attribution table. For output point attribution, the attribution table must include a map object table (.map).
- **Column Value attribution** – matches (joins) a column in the geocoded table to an equivalent column in the attribution table. The column in the attribute table must be indexed.

Attribution operates during a geocoding pass or as a separate batch process. Only successfully geocoded or matched records receive attribute data. Batch attribution generally takes less time to complete than geocoding with attribution enabled. Column value attribution generally takes less time to complete than output point attribution.
General Attribution Process

The general process for attribution in MapMarker is as follows:

1. Create output columns in the geocoding table to contain the attribute values (see Preparing Your Table for Attribution on page 86).
2. Select the attribute table (see Selecting an Attribution Table on page 87).
3. Link the attribute table to your geocoding table, either by output point (see Setting Up Output Point Attribution on page 88) or column value (see Setting Up Column Value Attribution on page 89), then bind together the attribute column in your table and the output column in the attribution table.
4. Geocode the table to add the attributes (see Adding Attributes Through Geocoding on page 89) or run the batch attribution process (see Batch Adding Attributes on page 90).

Preparing Your Table for Attribution

Before you can add attributes to your table, you need to create a new column to contain the attributes. To do this:

1. Open the MapInfo table to which you want to add attributes.
2. In the Select input columns from your table dialog box, click the MODIFY COLUMNS button.
   The Modify Table Structure dialog box displays.

3. Click the ADD COLUMN button to add a new column to the table. The Columns list box displays a new default 10-character column named Field# (where # is the column number in the table).
4. In the COLUMN INFORMATION group, define the NAME, TYPE, and WIDTH of the new column that will contain the new attributes. This column should have the same type and width as the column in the attribution table you will be pulling attributes from. Click OK when done editing the new column’s information.

   A prompt will display, informing you that the table will need to be closed and reopened to complete the update. Click YES. The table is updated with the new column and the Select input columns from your table dialog box displays.
5. Finish selecting your input columns as needed and then click **NEXT**.

6. In the Select output columns from your table dialog box, select your output columns and then click **FINISH**.
   
   If the **GEOCODE** dialog box displays, click **CANCEL**.

Your table now contains a blank column that will contain the attributes from your attribution table. You can view the added column using the table browser.

**Selecting an Attribution Table**

Once you have added an attribute column to your table (see *Adding Attributes to Your Table on page 89*) you can then begin setting up your attribution preferences, starting with selecting the attribution table you want to pull attributes from.

To select an attribution table:

1. On the **TABLE** menu, point to **ADD ATTRIBUTES** and click **SELECT ATTRIBUTE FILES**. The Select Attribute Tables dialog box displays.

2. Navigate to the location of the attribute table you wish to pull attributes from. Use the **DRIVES** drop-down list to select the drive containing the attribute table, and then navigate that drive using the **AVAILABLE TABLES** list box.

3. When you locate the attribute table you wish to use, click it and then click **ADD**. The full path to the table displays in the **SELECTED TABLES** box. Repeat to add additional tables.

4. When finished selecting attribute tables, click **OK**. The Bind Attribute Table Columns to Output Columns dialog box displays.
You are now ready to bind the attribute columns in your working table to attribution table columns. If you want to add attributes based on common geography, see Setting Up Output Point Attribution on page 88. If you want to add attributes using joined columns, see Setting Up Column Value Attribution on page 89.

You can perform both types of attribution at the same time, provided you have selected the appropriate types of tables to use. Each attribution table you select can only be used for one type of attribution, either output point or column value.

Setting Up Output Point Attribution

Output point attribution links the geocoded table to the attribution table by geography, specifically by matching latitude and longitude point coordinates within the geocoded table to a geographic boundary defined within the attribution table. For output point attribution, the attribution table must include a map object file (.map).

To set up output point attribution:

1. After selecting your attribute table(s) (see Selecting an Attribution Table on page 87), select the table from which you want to copy attributes from the ATTRIBUTE TABLE NAME drop-down list on the Bind Attribute Table Columns to Output Columns dialog box.
2. Click the OUTPUT POINT option in the JOIN USING group.
3. In the input table list box at the bottom left of the dialog box, click the column in your table into which you want to add attributes.
4. In the output table list box at the bottom right of the dialog box, click the column in the attribute table from which you want to add attributes.
5. Click the BIND TO COLUMN button. The input column list box indicates the output column number to which the input column is bound.

Repeat this process for each output point attribution you wish to perform. For each table you select, you can choose whether to perform output point or column value attribution.

6. When finished, click OK.
Setting Up Column Value Attribution

Column value attribution matches (joins) a column in the geocoded table to an equivalent column in the attribution table. The column in the attribute table must be indexed.

To set up column value attribution:

1. After selecting your attribute table(s) (see Selecting an Attribution Table on page 87), select the table from which you want to copy attributes from the Attribute Table Name drop-down list on the Bind Attribute Table Columns to Output Columns dialog box.
2. Click the COLUMN VALUE option in the JOIN USING group.
3. In the Attribute Table Column list box, click the column in the attribute table that you want to join to the output table.
4. In the Output Table Column list box, click the column in the output table that you want to join to the attribute table.
5. In the Characters Used in Comparison box, type the number of characters you want MapMarker to use in determining a match between the joined attribute and output columns.
6. In the input table list box at the bottom left of the dialog box, click the column in your table into which you want to add attributes.
7. In the output table list box at the bottom right of the dialog box, click the column in the attribute table from which you want to add attributes.
8. Click the Bind to Column button. The input column list box indicates the output column number to which the input column is bound.

Repeat this process for each column value attribution you wish to perform. For each table you select, you can choose whether to perform output point or column value attribution.

9. When finished, click OK.

Adding Attributes to Your Table

Once you have bound your attribute and output table columns (see Setting Up Output Point Attribution on page 88 and Setting Up Column Value Attribution on page 89), you are ready to add attributes to your table. One means of performing attribution is through geocoding your table (see Adding Attributes Through Geocoding on page 89). You can also opt to add attributes using a batch process (see Batch Adding Attributes on page 90), which is performed without geocoding the table. Output point attribution must be performed on a table either during a geocoding run or by a batch process on a table that has already been geocoded. Column value attribution can be performed prior to, during, or after geocoding.

Adding Attributes Through Geocoding

When you geocode the table, each record in your table is being opened and processed. At this time, MapMarker also pulls the attributes defined for your record from the attribute table columns you specified while setting up attribution and inputs those attributes into your table’s target column(s).

For information on geocoding your table, see Geocoding with MapMarker on page 31.
Batch Adding Attributes

You can batch add attributes to your table any time after setting up attribution. When you batch add attributes, MapMarker uses the table joins and bindings you defined when setting up attribution, and copies the matched attributes from the attribute table(s) and adds them to your table in the column(s) you specified.

To batch add attributes:

- On the **TABLE** menu, point to **ADD ATTRIBUTES** and click **BATCH ADD ATTRIBUTES**.
  The Updating Table process indicator displays and automatically closes once attribution is complete.

Stopping Attribution

Once attribution is set up for your table, MapMarker will perform attribution on your table every time you geocode until you instruct it to stop. Attribution will be enabled every subsequent time you open the table as well, until you manually stop attribution for the table. Stopping attribution does not remove the attribution information for the table, rather, it stops any further attribution for the table while it is open in MapMarker. To completely remove all attribution information for the table, see **Clearing Attribute File Selections on page 90**.

To stop table attribution:

- On the **TABLE** menu, point to **ADD ATTRIBUTES** and click **STOP ATTRIBUTION**.
  When you stop attribution, MapMarker will no longer perform attribution on the table while the table is open, but the attribution settings will still be defined within your table’s metadata. You can view the metadata by opening the *.tab file in a text editor.

  The attribution information is located at the bottom of the metadata, and resembles the following:

  ```
  \"\MapMarker\AddAttributes\" = \"\n  \"\MapMarker\AddAttributes\InputTables\" = \"1\n  \"\MapMarker\AddAttributes\Layer0\" = \"c:\program files\mapinfo\mapmarker\sampdata\cntydemo.tab\n  \"\MapMarker\AddAttributes\Layer0\BindOnPoint\" = \"1\n  \"\MapMarker\AddAttributes\Layer0\Attribute0\" = \"Ctyname County_nam\n  \"\MapMarker\AddAttributes\Layer0\AttributeCount\" = \"1\n  ```

Clearing Attribute File Selections

If you decide that you no longer want to perform attribution for your table, you can clear the attribute file selections for your table. The table will no longer be set up for attribution. When you clear attribute file selections, all attribution information in your table’s metadata is removed permanently. If you want to perform attribution on your table in the future, you must configure new attribution settings for your table (see **General Attribution Process on page 86** for the basic steps to setting up and performing attribution.)

To clear attribute file selections:

- On the **TABLE** menu, point to **ADD ATTRIBUTES** and click **CLEAR FILE SELECTIONS**.
  If you want to stop the attribution process without removing the attribution metadata from your table, see **Stopping Attribution on page 90**.
Using the Sample Attribute File

Before trying attribution out on your own tables, we suggest you use the sample files provided with MapMarker in order to see how attribution works and how it can be a useful addition to your geocoding process. We provide a sample attribution table in the \MapMarker\SAMPDATA directory called CNTYDEMO.tab. Use this table to try out the attribution feature using Output Point attribution. Follow the steps below:

1. Open US_ADDR.tab from within the SAMPDATA folder in your product installation directory.
2. In the Select input columns from your table dialog box, click the MODIFY COLUMNS button.
3. Click the ADD COLUMN button to add a new column to the table. The COLUMN INFORMATION group should display information for a new character column named Field23 with a width of 10.
4. Change the new column’s name to County_Name and change the width to 30 characters, then click OK.
   A prompt will display, informing you that the table will need to be closed and reopened to complete the update. Click YES.
   The table is updated with the new column and the Select input columns from your table dialog box displays.
5. Click NEXT, then in the Select output columns from your table dialog box click FINISH. If the GEOCODE dialog box displays, click CANCEL.
6. On the TABLE menu, point to ADD ATTRIBUTES and click SELECT ATTRIBUTE FILES. The Select Attribute Tables dialog box displays.
7. Navigate to the SAMPDATA folder within your product installation directory, select CNTYDEMO.tab, and then click ADD. The path to the file displays in the SELECTED TABLES box; click OK. The Bind Attribute Table Columns to Output Columns dialog box displays.
8. Click the OUTPUT POINT option in the JOIN USING options group. In the left attribution box, choose the CTYNAME column, and in the right attribution box choose the COUNTY_NAME column.
9. Click the BIND TO COLUMN button and then click OK.
10. On the TABLE menu, click GEOCODE. The Geocode dialog displays.
11. Click the ALL TYPES option in the TYPE OF ROWS TO GEOCODE option group and then click the GEOCODE button to geocode the US_ADDR.tab with your preferred settings.
12. At the conclusion of the geocoding pass, look at the County_Name column in the table browser to see the attributes applied from CNTYDEMO.tab.
   Only records that were successfully geocoded have attribution output.
Attribute Considerations

Please consider the following when using Attributes.

- The **ADD ATTRIBUTES** menu group is available only after you have selected the input and output columns for your geocoding table.
- Do not open the attribute table in MapMarker. As long as you provide the path in the Select Attribute Tables dialog box, MapMarker is able to retrieve information from it.
- Attribution can be performed on any local table that MapMarker can geocode. It can also be incorporated into a batch geocoding session, for example, for overnight processing. See **Batch Geocoding on page 83**, for instructions on setting up a batch operation.
- MapMarker does not output attributes to columns that have already been selected in the Select input columns from your table and Select output columns from your table dialog boxes that display when opening a table in MapMarker.
- For a column value match, the matching column in the attribute table must be indexed. In addition, the values in the columns must be an exact match. This can be controlled by defining the number of characters required for an exact match in the Bind Attribute Table Columns to Output Columns dialog box. See **Setting Up Column Value Attribution on page 89** for more information.
- For output point attribution, the attribute table must include a map object file (.map). The table you are geocoding must contain longitude and latitude values, not just the point object. The coordinates can be obtained during the same geocoding/attribution pass. See **Setting Up Output Point Attribution on page 88** for more information.
- For proper storage of attributes, keep in mind the type and length of attribute fields you are working with. The column type/length for the attribute and output columns display in the Bind Attribute Table Columns to Output Columns dialog box. For example, a 15-character value in the attribute table is truncated if the output column can only accept 10 characters. Character values stored as numerics in the attribute table may not display as expected in the output table.
- If MapMarker finds a duplicate match in the attribution table, it takes the attributes from the first record it finds.
In addition to geocoding MapInfo native format (.tab) and dBASE (.dbf) tables, MapMarker can reach a wide variety of remote file formats to spatially enhance your business data. MapMarker geocodes remote relational databases via ODBC, an open database connectivity standard. While the general process of geocoding these databases is the same as for local tables, there is enough information specific to remote table geocoding that warrants a separate discussion.

MapMarker can geocode tables that are stored on remote database systems in addition to local tables. MapMarker uses ODBC to access the data and store the resulting geocoding data and spatial object remotely. This ensures that data integrity and security are maintained.

In this chapter:

- Supported Databases ................................................................. 94
- Remote Tables and Spatial Objects .............................................. 94
- Preparing to Geocode Remote Databases .................................... 95
- Accessing a Remote Table for Geocoding ................................... 96
- Writing Geocoding Output to Another Table ............................... 98
- Additional Considerations for Remote Table Geocoding ............... 102
Supported Databases

MapMarker and MapMarker Plus support the following databases:

- Oracle 10G
- SQL Server 2000
- MS Access 2000

The following drivers are supported:

- Oracle 9.2.06
- SQL Server 2000.81.9001
- MS Access 4.00.5303.01

The SQL Server and MS Access drivers are installed with MDAC 2.7.

You must install the Oracle driver separately. To use the driver, you must have the Oracle 9i client installed.

Remote Tables and Spatial Objects

MapMarker supports remote database geocoding via ODBC for relational database management systems (RDBMS). The RDBMS stores coordinates in X,Y columns. The RDBMS stores the geocoded spatial object directly.

Mappable Tables and the MapInfo Map Catalog

This section introduces you to mappable tables and the MapInfo Map Catalog, features that are an integral part of remote geocoding with MapMarker. Remote mappable tables and the MapInfo Map Catalog were first introduced in MapInfo Professional as a way to communicate spatial information about remote data to the MapInfo product line.

A mappable table is a table of data that has its spatial information recorded in the MapInfo Map Catalog. Mappable tables are managed by the MapInfo Map Catalog. MapMarker reads the Map Catalog each time you geocode a remote mappable table to know how to handle the spatial object.

A Map Catalog is optional for X,Y databases. In every Map Catalog an entry is required for each table in the database that you wish to geocode.

We recommend that you create your Map Catalog and make a table mappable within MapInfo Professional. Specific tools and commands are provided there for this purpose.

If, however, you do not have access to MapInfo Professional, you can create these two procedures manually. See Appendix D: Creating a Map Catalog By Hand, for specific instructions.

Supported Spatial Index Types

MapMarker currently supports X,Y (for Oracle, SQL Server, and MS Access) represented in the Map Catalog under the column SPATIALTYPE for X,Y.
Preparing to Geocode Remote Databases

Before you can geocode your remote table with MapMarker, you and/or the database administrator must set up MapMarker and your database to handle the geocoding operation. In general, the process follows these steps:

1. Create a data source for the RDBMS (see Creating a Data Source on page 95).
2. Prepare the remote table with the appropriate columns and spatial handling requirements (see Preparing the Remote Table on page 95).
3. Create a Map Catalog for each remote database (see Creating a Map Catalog on page 95).
4. Make the remote table mappable (see Making a Remote Table Mappable on page 96).

Creating a Data Source

After the drivers are installed, create a data source for your database. Use the ODBC Administrative Tools in the Control Panel for this purpose. Follow the steps below:

1. In Control Panel or Control Panel>Administrative Tools, double-click on 32BIT ODBC to display the ODBC DATA SOURCE ADMINISTRATOR dialog box. Under the USER DSN tab, click ADD.
2. At the CREATE NEW DATA SOURCE dialog box, choose the appropriate driver. Click FINISH. The ODBC SETUP dialog box is displayed. The dialog box differs depending on your database.
3. Fill out the dialog box per your needs and click OK. The SELECT button allows you to associate a specific database with the data source.
4. Return to the ODBC DATA SOURCE ADMINISTRATOR dialog box and click OK.

Preparing the Remote Table

You or your database administrator must add columns to the table that stores the results of the geocoding operation. For the X,Y spatial index type, these columns include one each for longitude and latitude values. Each should be of decimal (11,6 or larger) or double. Refer to Output Column Type/Width Requirements on page 33.

In addition, you may wish to add a column to your table for storing the result code that MapMarker generates, which describes the type of geocoding match that was attempted.

If you plan to write geocoding output to another table, provide those columns as well. See “Writing Geocoding Output to Another Table section on page 98” for more information.

Creating a Map Catalog

Create a Map Catalog for each remote database you want to access in MapMarker. In MapInfo Professional, run the utility MIODBCAT.MBX from the RUN MAPBASIC PROGRAM command. For complete details see the MapInfo Professional User’s Guide, Chapter 21: Accessing Remote Database Data.
If you are not a MapInfo Professional user, you must create the Map Catalog manually, as described in Appendix D: Creating a Map Catalog By Hand. This process only needs to be performed once for each database.

Making a Remote Table Mappable

For each spatial table in the remote database that you want to access in MapMarker, you must include it as a row in the MAPINFO_MAPCATALOG table. This is carried out in MapInfo Professional when you choose TABLE>MAINTENANCE>MAKE ODBC TABLE MAPPABLE. For details, see Chapter 23 in the MapInfo Professional User’s Guide.

If you do not use MapInfo Professional to manage the Map Catalog, you must manually add rows to the MAPINFO_MAPCATALOG table for each spatial table in the database that you want to geocode. See Appendix A for further instructions.

The browser below illustrates a portion of the Map Catalog for a database with four mappable tables. The coordinates for these tables are stored in X,Y columns, as indicated by the SPATIALTYPE 4.

Accessing a Remote Table for Geocoding

Now that your remote database is prepared for geocoding, you are ready to open it in MapMarker and begin geocoding. Follow the steps below.

1. If you have specified that the File Open dialog box display at startup for ODBC tables (set in SYSTEM PREFERENCES), the SELECT DATA SOURCE dialog box is displayed automatically when you start MapMarker.

   Otherwise, choose FILE>OPEN ODBC TABLE or click to display the SELECT DATA SOURCE dialog box.

2. Click the MACHINE DATA SOURCE tab.
3. Choose your data source. Your data source is the location of the specified database. For example, Access Files could be the name of the data source that provides access to one or more Access tables located in a specified directory. Once connected to the Access Files data source, you can access information from any Access table.

4. Sign on to your data source in the usual way. This procedure differs depending on what type of ODBC table you are using.

5. Choose the table you want to access from the SELECT TABLE dialog box. Each of the tables owned by you or accessible to you is displayed.

Note: Keep in mind you cannot open any other tables while you have an ODBC table open.

6. Proceed with geocoding as you would with any other MapInfo table (see Chapter 3: Geocoding Your Table Automatically and Chapter 3: Geocoding Your Table Interactively). The procedure is the same as geocoding local tables, with two exceptions, as explained below.

Setting the Datum for Remote Tables

For remote table geocoding, MapMarker only geocodes to an NAD 27 or NAD 83 datum. If your remote table uses another datum, MapMarker prompts you to choose NAD 27 or NAD 83 before proceeding.

MapMarker determines the datum for the table by checking the System Preferences setting and the value in the Projection column in the Map Catalog. If there is a conflict between the two, MapMarker prompts you to change the projection in the Map Catalog to agree with the System Preference setting. If you respond with No, the process is unable to continue. A message is displayed indicating the table is not mappable and cannot be geocoded.
In the Datum tab of the System Preferences dialog box, the options “Use NAD 83 for tables with no datum” and “Use NAD 27 for tables with no datum” act the same as if you chose one of the first two options, “Convert all tables to NAD 83” or “Convert all tables to NAD 27.”

Be sure that you use the same datum for the entire table to ensure consistency among geocoded records. For more on datums, see Understanding Datums in Appendix C on page 375.

Writing Geocoding Output to Another Table

MapMarker has the ability to geocode a remote database table and write the results to a related table. Often database administrators do not want users to modify or write back to the source table, because of the potential of introducing errors into the database. In that case the DBA can set up a related table to store the geocoding results and other output information. During geocoding MapMarker reads the input addresses from the source table and writes the results of the operation to the related results table.

Relationship Between Source and Results Tables

The source and results tables must be linked by a common key. In the source table, the values of the primary key must match the values in the unique key of the results table. We recommend relating the tables in a one-to-one relationship. Each address record in the source table has a corresponding record in the related results table connected by the key values.

Where there is more than one record with the same address (a multiple-to-one relationship), these records may produce georeresults that are written to one record. Each time MapMarker geocodes another address with the same value, it updates the output record in the results table. As long as each address record with the same key contains the same address, the output is identical.
However, if two source records with the same key contain different address information, MapMarker geocodes the first record, writes its results, geocodes the second record, and updates the results with different information based on the different, and possibly incorrect, input address.

If your source table contains multiple unique keys, MapMarker prompts you to choose which keys are linked to the keys in the results table.

Setting Up for Related Table Geocoding

Prior to geocoding, the DBA should create a table for geocoding results that is related to the source table. The table structure should include at least one key that uniquely identifies an address in the source table. In addition it should, at least, contain fields for the geocoding results, and longitude and latitude coordinates. Other optional output fields include firm, street, city, ZIP Code, ZIP+4, Carrier Route, Delivery Point, Check Digit, and Census Block ID. See Output Address Columns on page 32, for details on column type and width. Consult your database documentation for information on creating relational tables and primary keys.

The DBA must also create a data source for the remote source/results tables. The data source should include information for MapMarker to determine which data to access and how to access it. See Preparing to Geocode Remote Databases section on page 95, for the instructions for setting up a data source.

Process for Remote Table-to-Table Geocoding

Use this procedure to geocode a remote table and write the results to a related table:

1. In MapMarker, choose File>Open ODBC Table.
2. Choose the data source and enter your user name and password if necessary.
3. Choose the source table that contains the addresses you want to geocode. Enter the password if your data source requires it.
4. For writing georeresults to a related table, answer Yes when prompted.
5. Choose the data source for the output table and log on with your user name and password.
6. Choose the results table that will contain the geocoding output. Click OK. If your source table has multiple unique keys, the Describe Table Relationship dialog box is displayed. Proceed to step 7. If your table has only one key, the Select Input Column dialog box is displayed. Skip to step 9.
7. In the Describe Table Relationship dialog box, choose the address source columns and the corresponding columns with unique keys in the results table. In the Input Table list, select the input column and drag it to the Selected Input Columns box for Key 1. In the Output Table list, select the input column/key that is linked to the input column and drag it to the Selected Output Keys box for Key 1. Repeat for additional columns and keys. To remove a column name, drag the <Clear> item on top of them.
8. Click **OK**. The **Select Input Column** dialog box is displayed.

9. In the **Select Input Columns** dialog box, choose the columns that contain the appropriate address components. Click **Next**. The **Select Output Columns** dialog box is displayed.

10. Identify the columns in the results table where MapMarker should store the result codes, coordinates and output addresses.

11. Click **Finish** to close this dialog box. The **Geocode** dialog box is displayed if set to display automatically in System Preferences. If not, choose **Table>Geocode**.

12. In the **Related Table** tab, choose to optimize MapMarker for inserting records in the remote table or for updating existing records.

13. Continue with the normal geocoding process by choosing from among its many options such as automatic or interactive geocoding, geocoding to street addresses or ZIP Code centroids etc. See Chapter 3: Geocoding with MapMarker, for more information.

**Note:** When geocoding using multiple tables, the options, “Georesult less than” and “Unmatched rows” are not available. The option, “All types” is available.
Optimizing MapMarker for Related Table Geocoding

You may be writing the geocoding results to a new output table or are updating previously geocoded records in an existing related table. In order to get the best performance from MapMarker during this operation, you can set it in favor of one situation over the other. In the GEOCODE dialog box, choose from two settings in the Related Table tab: Optimize for Insertion or Optimize for Update.

As a rule of thumb, choose **Optimize for Insertion** when the results table is empty or nearly so. **Choose Optimize for Update** when the results table already contains many records that will likely be updated. The default setting is Insertion.

**Note:** This feature does not apply to geocoding to a table in a remote database where the input and the output results are stored in the same table. In these instances, the Related Table tab is not visible in the GEOCODE dialog box.

Remote Database Views

MapMarker enables you to geocode views of remote tables provided you are writing the geocoding output to another table. A view is a subset of a table that is typically the result of a select statement. For example, you may create a view of a nationwide table that includes only records for California. Refer to your database documentation when creating views.

Once your view is created and you are ready to geocode, follow these steps:

1. Choose **File>Open ODBC Table**. The **Data Source** dialog box is displayed.
2. From the **Machine Data Source** tab, choose the data source for the view. Click **OK**.
3. From the **Select Table** dialog box, click the **Filter** button. Select the **Views** check box and click **OK**. The **Select Table** dialog box is displayed again with a list of views for the data source.
4. Choose the view that represents the input table and click **OK**.
5. Answer **Yes** to using a different table for the outputs from geocoding.
6. From the **Select Table** dialog box, choose the view for the output table and click **OK**.
7. At the **Select Input** and **Output Column** dialog boxes, choose your columns and click **OK**.
8. From the **Table>Geocode** dialog box, proceed with the normal geocoding process by choosing from among its many options such as automatic or interactive geocoding, geocoding to street addresses or ZIP Code centroids, and setting matching criteria. See Chapter 3: Geocoding with MapMarker, for more on the geocoding process.

Related Table Considerations

MapMarker’s ability to write geocoding results to another table is only available for remote related tables accessible through MapMarker’s ODBC support. All addresses must be contained in the source table, while all results must be written to a related results table.

You or your DBA must be familiar with how the two tables are related. If the tables use multiple keys, you need to set the relationship before starting any geocoding. Be familiar with your data and table structure to ensure the geocoding output results are valid.
Additional Considerations for Remote Table Geocoding

The following should be considered when performing any type of remote table geocoding.

Sort by ZIP Code for Faster Geocoding

MapMarker’s performance is best when it can access records sorted by ZIP Code. For local tables, therefore, we recommend that all tables for geocoding are sorted by ZIP Code prior to geocoding. For remote tables, we recommend that the ZIP Code column be indexed for faster accessing. To add an index to a remote table, follow the instructions for your database.

Geocoding Unmatched Records

MapMarker can geocode unmatched records in previously geocoded remote tables instead of geocoding the entire table with every pass. After you select the input and output columns, go to the GEOCODE dialog box and set the types of rows to geocode to Unmatched Rows. MapMarker attempts to find a match for all records in your table that have a geosresult of N or blank. MapMarker updates the Log file to reflect the total number of records it attempted to geocode during the pass, not the total number of records in the table.

Note: Geocoding unmatched records is not available for multi-table geocoding.

Transaction Control with ODBC Tables

MapMarker uses as non-obtrusive locking mechanism as possible when geocoding your ODBC table. It attempts to use the lowest level of locking available (e.g., row, page or, at worst case, table locking). Locks are not placed until the row is geocoded. MapMarker issues a commit after each update so that each lock is held for an extremely short time (the time it takes to lock a row, update it, and commit it).

If MapMarker attempts to lock a row that is already locked, MapMarker follows the default behavior of the database. Usually this means waiting until that row is released. MapMarker does not use any deadlock detection beyond that provided by the database.

Note: You are responsible for maintaining the relationship of the street address and the latitude/longitude for that address. For example, whenever you change any street address information, you should re-geocode that record. For Informix databases, MapInfo provides a MapInfo Geocoding DataBlade based on MapMarker technology that automatically geocodes rows as they are inserted or updated.

Finally, if rows are deleted from your table after MapMarker opens the table, MapMarker reports an error when it tries to geocode that deleted row.

Remote Database Views

You may only geocode views of a remote table when you are performing a table-to-table geocode. See the Writing Geocoding Output to Another Table section on page 98, for more information.
Rollback Segment Limit

Be sure that you set your database rollback segment or temp space to be sufficiently large enough to accommodate the size of your database.

Supported Unique Index and Primary Key Data Types by Database

MapMarker supports ODBC tables in MS Access 2000, Oracle10G, and SQL Server 2000. The following table identifies the supported data types by RDBMS. A Primary Key must also be defined for your table.

<table>
<thead>
<tr>
<th></th>
<th>MS SQLServer 7.0</th>
<th>MS Access</th>
<th>Oracle10G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char</td>
<td>Character</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>Float</td>
<td>Number (double)</td>
<td>Number</td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>Number (Long Integer)</td>
<td>Varchar</td>
<td></td>
</tr>
<tr>
<td>SmallInt</td>
<td>Number (Integer)</td>
<td>Varchar2</td>
<td></td>
</tr>
<tr>
<td>Tinyint</td>
<td>Number (Single)</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Varchar</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

If MapMarker encounters a mappable table with a spatial type it does not support, an error message is displayed and the table is not geocoded.
This chapter explains how to create and manipulate a custom address dictionary. It also explains how you can use the place names in the user dictionary as geocoding input.

In this chapter:

- Creating a Customized User Dictionary ........................................... 106
- User Dictionary Considerations ......................................................... 109
- Geocoding to Place Names ............................................................. 111
Creating a Customized User Dictionary

MapMarker supports the creation and use of customized dictionaries as a supplement to the MapMarker Address Dictionary. A customized dictionary is a table of streets and address ranges that you match ungeocoded records against. You can use more than one custom dictionary when geocoding; however, you are limited to five dictionaries at one time.

**Required Fields**

To create a user dictionary, start with a MapInfo table of street segments or points that contains addresses you want to geocode against. Your MapInfo table must contain certain fields, which MapMarker then uses to convert the table into a dictionary format. The table below lists the required fields.

<table>
<thead>
<tr>
<th>Required Fields</th>
<th>Description</th>
<th>Maximum Field Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Start Address</td>
<td>Start of address range on left side of street</td>
<td>10</td>
</tr>
<tr>
<td>Right Start Address</td>
<td>Start of address range on right side of street</td>
<td>10</td>
</tr>
<tr>
<td>Left End Address</td>
<td>End of address range on left side of street</td>
<td>10</td>
</tr>
<tr>
<td>Right End Address</td>
<td>End of address range on right side of street</td>
<td>10</td>
</tr>
<tr>
<td>Street Name</td>
<td>Name of street</td>
<td>30</td>
</tr>
<tr>
<td>State Abbreviation</td>
<td>2-character state abbreviation</td>
<td>2</td>
</tr>
<tr>
<td>Left ZIP Code</td>
<td>ZIP Code for left side of street</td>
<td>5</td>
</tr>
<tr>
<td>Right ZIP Code</td>
<td>ZIP Code for right side of the street</td>
<td>5</td>
</tr>
</tbody>
</table>

**Optional Fields**

You can also add a number of optional fields to your customized dictionary. These fields are listed in the table that follows.

The Odd/Even indicator fields, although not expressly required to create a user dictionary, may be needed depending on the data in your table. If your table contains odd/even indicator information, we strongly urge you to use the Odd/Even indicator fields. Use of these fields ensures that your geocoded addresses are located on the correct side of the street. Omitting the fields when your data contains odd/even information may produce incorrect results. If you do not have odd/even indicator information in your data, you can safely omit the fields. For more information, see *Odd/Even Indicator Fields on page 110*.
User Guides Chapter 8: User Dictionaries

MapMarker Plus 10.0

© 2004 MapInfo Corporation. All rights reserved.

Note that MapMarker only uses a user dictionary if you have licensed the areas covered by that dictionary. For example, if you create your own New York City user dictionary, you must have a license file for New York state or the entire United States.

The more user dictionaries you use, the more geocoding performance degrades. You are limited to using five user dictionaries at one time. If you need to use more than five, use Append.mbx (provided with MapMarker) to append some of your MapInfo tables together. Append.mbx is found in the MapMarker program directory. TheAppend utility is discussed in Appendix F: MapMarker Utility Programs. Create user dictionaries for smaller areas for best performance.

MapMarker provides a wizard to walk you through the user dictionary creation process.

To create a user dictionary:

1. Choose File>CREATE USER DICTIONARY. The USER DICTIONARY WIZARD (STEP 1 OF 3) dialog box is displayed.

<table>
<thead>
<tr>
<th>Optional Fields</th>
<th>Description</th>
<th>Maximum Field Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Odd/Even indicator</td>
<td>Left side of the street contains only odd or even address ranges (O=odd, E=even, B=blank)</td>
<td>1</td>
</tr>
<tr>
<td>Right Odd/Even indicator</td>
<td>Right side of the street contains only odd or even address ranges (O=odd, E=even, B=blank)</td>
<td>1</td>
</tr>
<tr>
<td>City</td>
<td>City name</td>
<td>28</td>
</tr>
<tr>
<td>Left ZIP+4 Code</td>
<td>4-digit ZIP+4 add-on for left side of street</td>
<td>4</td>
</tr>
<tr>
<td>Right ZIP+4 Code</td>
<td>4-digit ZIP+4 add-on for right side of street</td>
<td>4</td>
</tr>
<tr>
<td>Left Census Block</td>
<td>Census Block ID for left side of street</td>
<td>15</td>
</tr>
<tr>
<td>Right Census Block</td>
<td>Census Block ID for right side of street</td>
<td>15</td>
</tr>
<tr>
<td>Place Name</td>
<td>Place name</td>
<td>40</td>
</tr>
</tbody>
</table>

Optional Fields Description

| Maximum Field Length |

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Odd/Even indicator</td>
<td>Left side of the street contains only odd or even address ranges (O=odd, E=even, B=blank)</td>
</tr>
<tr>
<td>Right Odd/Even indicator</td>
<td>Right side of the street contains only odd or even address ranges (O=odd, E=even, B=blank)</td>
</tr>
<tr>
<td>City</td>
<td>City name</td>
</tr>
<tr>
<td>Left ZIP+4 Code</td>
<td>4-digit ZIP+4 add-on for left side of street</td>
</tr>
<tr>
<td>Right ZIP+4 Code</td>
<td>4-digit ZIP+4 add-on for right side of street</td>
</tr>
<tr>
<td>Left Census Block</td>
<td>Census Block ID for left side of street</td>
</tr>
<tr>
<td>Right Census Block</td>
<td>Census Block ID for right side of street</td>
</tr>
<tr>
<td>Place Name</td>
<td>Place name</td>
</tr>
</tbody>
</table>
2. Specify the path and filename of the MapInfo table from which address information is to be derived. Alternatively, click the BROWSE button to reach the desired location of the table.

3. Specify how much of the table you wish to include in the dictionary by specifying the row numbers or leave All Rows selected.

4. Click NEXT to continue. The USER DICTIONARY WIZARD (STEP 2 OF 3) dialog box is displayed.

5. Specify the required fields in the street table by highlighting a field name in the list box and dragging it to the appropriate box in the address groups. Note that you must fill in all the fields in the dialog box.

When selecting the state field choose the field that contains the two-letter state abbreviation. Do not use the 2-digit FIPS code field as the state field.

6. To clear any box, highlight <CLEAR> in the list and drag it over the column name you wish to delete.

7. Click NEXT to continue. The USER DICTIONARY WIZARD (STEP 3 OF 3) dialog box is displayed.
8. Specify any optional fields for City, ZIP+4, Census Block, Odd/Even Indicator, or place name columns you wish to include in the dictionary. Including the Odd/Even Indicator allows for better placement of address points when geocoding. A place name is a point object, rather than a street segment. Examples include Sears Tower, Wrigley Field, or City Hall.

9. Click Finish to proceed or Back to revisit the previous dialog boxes. When you click Finish the Save User Dictionary As dialog box is displayed.

10. Specify the location of the user dictionary. Note that it must be in a different location than the MapInfo source table.

11. Choose Options>System Preferences>Dictionary. The Dictionary dialog box is displayed. Select the User Dictionary check box and provide the path and filename in the content box.

   Note: The maximum length for a User Dictionary path is 256 characters. If you are using multiple user dictionaries, the total length of all paths cannot exceed 256 characters.

You can geocode with the MapMarker Address Dictionary and your custom user dictionary at the same time.

   Note: You cannot geocode to CASS standards with a user dictionary. The Use User Defined Dictionary option in the Dictionary dialog box is unavailable.

Address Range Order

MapMarker determines the order of the address range based on a comparison of the start and end addresses. The comparison produces the following results:

- if the end is greater than the start, the range is ascending;
- if the start is greater than the end, the range is descending;
- if the start is equal to the end, the range is ascending.

User Dictionary Considerations

Please consider the following when working with User Dictionaries.

Data Access License

You must still have a valid access license to the data contained in the MapMarker Address Dictionary when you are geocoding against your customized dictionary. For example, if you create a dictionary of New York streets and addresses, you must purchase the New York or entire U.S. MapMarker Address Dictionary.

Dictionary Path

MapMarker requires that you set the dictionary path(s) for every address dictionary you wish to access during a geocoding pass. Go to the Options menu and choose System Preferences. In the User Dictionary group, select the User Defined Dictionary check box and specify the path in
the path box. If you wish to geocode to only your custom dictionaries, clear the check box in the upper Address Dictionary group. Paths to User Dictionaries or Address Dictionaries cannot exceed 256 characters.

Odd/Even Indicator Fields

The Left and Right Odd/Even Indicator fields are used to specify that the sides of the street segment contain only odd or even address ranges. When the indicator is specified but is not consistent with address numbers specified, the indicator is set to “Both”. When the indicator is not specified and address numbers are specified, indicator is set to “Both” unless the start and end address numbers are equal. In that case, the indicator is set to “Odd” if the address numbers are odd and to “Even” if the address numbers are even. If this indicator is not specified and both Start Address and End Address have values, MapMarker defaults the indicator to be “Both” (odd and even).

Geocoding with Custom and Purchased Address Dictionaries

When geocoding to both the MapMarker Address Dictionary and your custom user dictionary, the match with the highest score takes precedence, regardless of which dictionary it came from. If the scores are equal, priority is given to the match in the custom user dictionary.

To determine from which Address Dictionary the result came, the final letter in the result code for street address or ZIP Code matches indicates U for user dictionary or A for MapMarker Address Dictionary. Street intersection matches include a [U] in the output street if the match came from a custom user dictionary.

Street Intersections and Customized User Dictionaries

When geocoding to street intersections with a custom user dictionary, MapMarker cannot recognize the intersections if one or more of the segments that make up the intersection does not have an endpoint at the intersection. This can happen when you create the User Dictionary from a customized street table where intersections were not constructed with every segment ending with an endpoint at the intersection (Example 1).

If you create your user dictionary from TIGER 2000 street geometry, MapMarker recognizes the street intersections (Example 2).
Geocoding to Place Names

MapMarker can now geocode to place names in the user dictionary. Place names are generally point objects rather than street segments, where the "from" and "to" fields contain the same number. To enter a place name in the address dictionary perform the following:

1. If the Place Name is associated with a single address, then in the left **START ADDRESS** and left **END ADDRESS** enter the number associated with the street address, if any. In the right **START ADDRESS** and right **END ADDRESS** fields enter 0.
2. Fill in the rest of the required fields.
3. Enter the Place Name and any other optional fields you may have.

If this is a new entry and there is no geometry object associated with the place name, it geocodes as an **S1** or **S3** depending on the ZIP fields in the user dictionary.

The place name can be made to geocode as an **S5** using MapInfo Professional:

1. Add latitude and longitude columns (float or decimal (11,6)) to the user dictionary table and enter the lat/long corresponding to your point.
2. In MapInfo Professional select the record.
3. Choose **TABLE>CREATE POINTS** and select the latitude and longitude columns. Click **OK**.

![Create Points dialog box](image)

After the user dictionary is created, the place name geocodes to an **S5** match.

© 2004 MapInfo Corporation. All rights reserved.
Part II: MapMarker Developer Guide

Part II, the Programmer’s Reference Guide, contains application programming interfaces for developers to create client/server or stand-alone geocoding applications.

Developer Guide Topics:

- **Client/Server Geocoding** .......................................................... 115
  This chapter introduces you to the world of client/server geocoding with MapMarker Server and the Client Geocoding Control (OCX). Add the control to your own application to call the MapMarker Server for instant geocoding information for any U.S. address.

- **OLE Automation API** ............................................................... 129
  Is the ready-made OCX too limiting for your needs? This chapter explains the OLE Automation API where you can customize your own geocoding OCX for your client application.

- **OLE Automation API for ColdFusion** ...................................... 185
  This chapter explains the ColdFusion API where you can use ColdFusion to customize your geocoding application.

- **OLE Automation API – For ASP/VBScript** ................................ 201
  This chapter explains the ASP/VBScript API where you can use ASP/VBScript to customize your geocoding application.

- **MapMarker Server RPC API** ................................................... 229
  The MapMarker Server RPC API is for the C developer who wants to create geocoding applications that call MapMarker Server. This discussion assumes you are familiar with Remote Procedure Calls.

- **MapMarker GeoEngine API** .................................................... 265
  This is the C API for the MapMarker geocoding engine used for creating batch geocoding applications like MapMarker. Included are the function calls, programming notes, and sample code to assist you. The MapMarker geocoding engine and its API are packaged as a 32-bit Dynamic Link Library (mm32v10.dll).

- **Data Dictionary API** ............................................................... 337
  Add custom user dictionary creation to your geocoding applications. This API includes a single function GeoEngDDCreateUserDictionary that allows you to create user dictionaries from a table of MapInfo objects (lines, polylines, or points) that contain address information. The code for the interface is integrated into the MapMarker geocoding engine.
Custom Applications and CD-ROM Unlocking

As a developer of geocoding applications, keep in mind that clients of your applications need access to encrypted data to run MapMarker successfully. You may accommodate them in either of two ways:

- The developer unlocks the data and installs the custom application for the client.
- The client unlocks the data before the application is installed on the system.

In the first situation, you, as the developer, unlock the data by installing MapMarker and calling MapInfo for an access code. At this time one or more license files are created in \mapinfo\mapmarker\data. To deploy your custom application, you must install the MapMarker engine (mm32v10.dll), the Address Dictionary, the license files, and your application on your client’s system. The client’s and your serial number for MapMarker must be the same since your serial number is embedded into the license files at the time of unlocking. The serial number is stored in the Windows Registry at HKEY_LOCAL_MACHINE \SOFTWARE\MAPINFO\MAPMARKER\<version number>\USER\SerialNumber.

Note: In reference to MapMarker registry keys, <version number> indicates the major-release version of MapMarker, e.g., 10.0, even if you are running one of the point releases of that version, e.g., 10.1, 10.2.

In the second scenario, the client unlocks the data by obtaining an access code from MapInfo during the installation of the standard MapMarker product. At that time the license file(s) is installed on the client’s system. This gives the client access to MapMarker and the Address Dictionary. Then install your application on the client’s system. No verification of serial numbers is required since the data is already unlocked and in place.
MapMarker’s flexible geocoding environment extends itself beyond the single workstation to include client/server settings that serve a wide variety of geocoding needs. Everything from roadside assistance, where-is-the-nearest location, visual multiple listing services for real estate, and telecom call-before-you-dig applications can put the power of MapMarker geocoding to practical use.

MapMarker’s client/server tools include MapMarker Server (mmServe.exe) and MapMarker Geocoder Control (mapmarkr.ocx). This chapter discusses how to use these tools in client/server geocoding.

For developers who are interested in creating a custom OCX for use with MapMarker Server, we provide the OLE Automation API, upon which the MapMarker Geocoder Control is based. The API is presented in the following chapters.

In this chapter:

- **Client/Server Geocoding** .............................................. 116
- **MapMarker Server** ..................................................... 116
- **Running MapMarker Server on Windows Systems** ............... 117
- **Running MapMarker RPC Server on UNIX Systems** ........................ 119
- **MapMarker Geocoder Control** ...................................... 120
- **Adding MapMarker OCX to Your Client Application** ............. 121
- **Setting MapMarker Client Control Properties** .................. 123
- **Geocoding via MapMarker Geocoder Control** ....................... 124
- **Falling Back to ZIP Code Centroids** ................................ 125
- **MapMarker Geocoder Control Events and Methods** ............... 125
- **Client/Server Geocoding and the Internet** ......................... 127
Client/Server Geocoding

MapMarker comes with a server and client OCX that allows MapMarker to geocode records from multiple users using a single geocoding engine. The components include MapMarker Server and MapMarker Client Geocoder Control. MapMarker Server can be run as a Windows Service or as a Windows console application.

MapMarker Server and the Geocoder Control are installed and registered on your system after you perform a full MapMarker installation or upgrade. See Installation and Upgrade Procedures on page 15 for more information.

Installing MapMarker Server and Client Control Manually

If you need to install MapMarker Server manually, see Installing MapMarker Server Manually on page 26.

To register the OCX control manually, see Registering the OCX Manually on page 26.

If you cannot register the OCX, check to see if you have the following .dlls in your Windows\System directory:

- msvcrtd.dll
- msvcrtd40.dll

MapMarker Server

MapMarker Server is a layer on top of the MapMarker geocoding engine, and extends the engine's functionality by providing a Remote Procedure Call (RPC) facility and a queuing/multi-threading function. The RPC interface of the MapMarker Server allows it to receive via TCP/IP geocoding requests from remote clients (such as the MapMarker Geocoder Control). Since the geocoding engine handles one request at a time, the MapMarker Server queues multiple requests until they can be fulfilled by the geocoding engine.

The MapMarker Server can handle a maximum of 1,024 simultaneous requests. In a typical customer service environment, a request might consist of an address being sent to the MapMarker Server, displaying the candidate list and then choosing the best candidate.

Even though the request may span 20 seconds or even 20 minutes, the geocoding engine is only occupied for an extremely short period of time. Depending on your hardware and network environment, the geocoding engine geocodes as many as 25 to 100 records a second. So even though there is one geocoding engine, many users may be served "simultaneously."

By default MapMarker Server geocodes addresses to street level. It returns potential matches from the MapMarker Address Dictionary or from custom user dictionaries. If you wish to geocode to ZIP Code centroids only or to retrieve match candidates from a custom dictionary only, you must set these parameters (-ziponly,-udonly) in the NT Services dialog box or at the command line. You may specify only one of these options at a time.
The MapMarker Server RPC interface is defined in Chapter 13: MapMarker Server RPC API. Most applications do not have to use this interface. Instead, we recommend that you use the MapMarker Geocoder Control, as described in this chapter. Or you can modify the OCX using the OLE Automation API, explained in Chapter 10: OLE Automation API. Application development should be easier with the OCX since you probably have to use C or C++ to access the RPC interface.

**Note:** The current implementation of the MapMarker Server does not support the Browse or GetNextRange features of the MapMarker geocoding engine. If you need these functions, you should use the MapMarker GeoEngine API directly (see Chapter 14: MapMarker GeoEngine API).

---

### Running MapMarker Server on Windows Systems

To run MapMarker Server as a Windows Service or Windows console application, see the appropriate section below.

#### Windows Service

To configure MapMarker Server as a Windows Service:

1. From the Control Panel in Windows NT or **CONTROL PANEL > ADMINISTRATIVE TOOLS** in Windows 2000 or XP, click **SERVICES**. The Services dialog box is displayed.
2. Highlight the service called MapMarker Server.
3. Choose the **STARTUP** button. Configure the MapMarker Server to run as the user who installed MapMarker. Be sure the user has permission to start a service at startup time. Click **OK** to return to the **SERVICES** dialog box.
4. Optional: To set the server for geocoding to a custom user dictionary, in the Startup Parameters list box add **-udonly**. MapMarker does not initialize the MapMarker Address Dictionary. It only gets match candidates from the user dictionary.
5. Optional: To set the server for geocoding to ZIP Code centroids, in the Startup Parameters list add **-ziponly**. MapMarker Server only geocodes to ZIP Code centroids; it does not return street address candidates.

**Note:** You may set only one startup parameter, either **-udonly** or **-ziponly**.

6. Click **START**. The MapMarker Server service starts.
7. To leave the **SERVICES** dialog box, choose **CLOSE**.

MapMarker Server is now running in the background. All log information is sent to the Event Viewer (found in the Administrator) under **LOG>APPLICATION**.

When the application is started, the server sends the message, “The MapMarker Engine Initialized with Database Path = [c:\mapinfo\mapmarkr\data;d:\data].” If the data path is incorrect, you can change it in `regedit`. In Windows 2000/XP/NT 4.0 Registry Editor (`regedit.exe`), find the settings under **HKEY_LOCAL_MACHINE\SOFTWARE\MapInfo\MapMarker\<version number>\SYSTEM**.

**Note:** In references to the MapMarker registry keys, `<version number>` indicates the major-release version of MapMarker, e.g., 10.0, even if you are running one of the point releases of that version, e.g., 10.1, 10.2.
If you encounter a problem with the Name Service Provider that prevents MapMarker Server from running when your machine boots, modify the NT Services feature as follows:

1. In Windows NT Control Panel, double-click on **NETWORK**.
2. Under the Services tab, double-click on **RPC CONFIGURATION** and change the Name Service Provider to DCE Cell Directory Service.
3. In the **NETWORK ADDRESS** dialog box, enter the machine’s IP address. If the machine is not on a network, leave this box blank.

To stop the MapMarker service:

1. Open the **CONTROL PANEL > SERVICE** application and highlight MapMarker Server.
2. Click **STOP**.

To remove the MapMarker service from the Windows NT Service Control Manager:

- At the DOS prompt, type `mm_serve -delete`.

### Console Application

To start MapMarker Server as a console application on Windows 98/2000/XP or Windows NT:

- At the DOS prompt type `mm_serve -console -udonly` (or `-ziponly`).

Specify only `-udonly` or `-ziponly` to set MapMarker Server to geocode to a customer user dictionary or to match to ZIP Code centroids.

When the application is started, the server sends the message, “The MapMarker Engine Initialized with Database Path = [c:\mapinfo\mapmarker\data;d:\data].” If the data path is incorrect, you can change it in `regedit`. In Windows 98/2000/XP/NT 4.0 Registry Editor (`regedit.exe`), find the settings under `HKEY_LOCAL_MACHINE\SOFTWARE\MapInfo\MapMarker\<version number>\SYSTEM`.

**Note:** In references to the MapMarker registry keys, `<version number>` indicates the major-release version of MapMarker, e.g., 10.0, even if you are running one of the point releases of that version, e.g., 10.1, 10.2.

To stop the application:

1. From the console window, press Ctrl-C.
2. When the DOS prompt is displayed, type `exit`. The console window closes.
3. Alternatively, press Ctrl-Alt-Del and select `mm_serve` from the **CLOSE PROGRAM** dialog box.

Any error information that is generated when MapMarker Server is running is displayed in the console window. There is no log file when running MapMarker Server as a console application.

For details on MapMarker’s geocoding operations, refer to earlier chapters of this Product Guide.

### Geocoding Request Timed Out

If MapMarker Server receives more than 1,024 requests, (i.e., all 1,024 available threads are occupied with other requests waiting for the Server), it may block your geocoding request and cause your request to time out. At that time, MapMarker Server displays error 32104: Mutex timed out. In this case, simply resubmit your request.
This error may also occur if you have started two instances of MapMarker Server on the same machine. This commonly happens when you start the server as a Windows Service and as a console application. This error is displayed in the Event View log under LOG>APPLICATION.

Running MapMarker RPC Server on UNIX Systems

MapMarker RPC Server runs as a UNIX process that can be stopped and started.

Please keep in mind that MapMarker RPC Server for UNIX runs only on Solaris operating systems.

Starting MapMarker RPC Server

To start the MapMarker RPC Server:

- Properly configure your environment. This is normally done at the time of installation. No special user account is required.
- Ensure the set of environment variables are active to start MapMarker properly.
- Set a path to the MapMarker software.
- Have access to data dictionary files for MapMarker.

Keep in mind, command syntax may vary depending on the shell environment.

Use the UNIX command mm_server to start the server. It may be run as a background job as follows:

```bash
nohup mm_server &
```

The nohup parameter leaves the server running after you log off. This results in a display in the nohup.out file similar to the following:

MapMarker server is using environment (as specified or by default):

```text
MAPMARKER_DBPATH /hpdskm/data
MAPMARKER_DBNAME geo_usa
MAPMARKER_USER_DBNAME
MAPMARKER_LINEAR_OFFSET 25.000000
MAPMARKER_CORNER_OFFSET 20.000000
MAPMARKER_UNITS 0
MAPMARKER_RELAX_HOUSE TRUE
MAPMARKER_RELAX_STREET TRUE
MAPMARKER_RELAX_ZIP TRUE
MAPMARKER_EXACT_CITY FALSE
MAPMARKER_PREFER_UD FALSE

mm server is running.
```

While the content of the variables vary, the key is the last line, mm server is running. If this does not display, there is a problem in the environment that must be fixed.
**Shutting Down the MapMarker RPC Server**

The MapMarker RPC Server is a forking server, meaning that there may be multiple active child processes from the server running at any point. If you shut down the server while there are active processes, you may cause errors for those client-side RPC transactions using them.

Use the UNIX `kill` command to shut down the MapMarker RPC Server. Find the process ID for `mm_server` using the UNIX command `ps`, paying attention to the PID and PPIDs to distinguish the server from its child processes. For example, the following command kills the process for `mm_server` running as PID 2149.

```
kill 2149
```

**MapMarker Geocoder Control**

The MapMarker Geocoder Control is a ready-made OLE control object for MapMarker Server. In the simplest case, you can use the control without any programming. It is designed for any application programming environment that can call OCXs such as Visual Basic or C++. To use the MapMarker Geocoder Control drag and drop the control onto your form. When you run your application, the MapMarker control object appears as shown below. Through the control an end user is able to connect to the MapMarker Server and send geocoding requests.

MapMarker Geocoder Control uses the RPC API to communicate with MapMarker Server. The client control connects, geocodes and disconnects for each geocode request (i.e., when the user clicks the GEOCODE button). This ensures that the geocoding request grabs one of the MapMarker Server’s 1,024 threads for the shortest time period. This reduces the time other geocoding requests wait for a thread.

If you want more control over the MapMarker Geocoder Control or want to provide a different user interface you can drive the control via the MapMarker OLE Automation interface. Since the OCX uses the same OLE Automation interface, a direct user of the Automation interface has access to the same methods and properties of the Geocoder Control. The OLE Automation API is discussed in Chapter 10: OLE Automation API.
Adding MapMarker OCX to Your Client Application

When you install MapMarker Server, the MapMarker Geocoder Control is installed and registered as well. The file `mapmarkr.ocx` is located in `\Program Files \Common Files \MapInfo Shared\Common dlls\`.

With MapMarker Geocoder Control installed, you are ready to add it to your application. The steps are outlined below. Here we are using Visual Basic as the development environment, but any tool that supports OLE Automation can be used. This discussion assumes that you are familiar with developing OCX applications.

1. In Visual Basic, add MapMarker Geocoder Control to the Toolbox by choosing `PROJECT>COMPONENTS` or by right-clicking on the Toolbox and selecting `COMPONENTS`.

   ![Components](image)

   2. Select the **MapMarker Geocoder Control** check box. Click **OK**. The icon for the MapMarker control is displayed in the Toolbox.

   3. Highlight the MapMarker Geocoder Control icon and drag and drop it onto your design form, or double-click on the icon to add the control to the form.
4. To set the properties, right-click on the control. The PROPERTY PAGES dialog is displayed.

5. Set the match restrictions for house number, street name, ZIP Code, street intersection, and expanded search in the Match Restrictions tab. In the List Candidate group, specify whether you wish to get all candidates returned or only close matches.

6. Click on the Offset, Connection and General tabs to set additional properties as necessary.

7. Choose RUN>START or click the START button to run the application in debug mode.

8. If you wish to create an executable, choose FILE>MAKE EXE FILE.
Setting MapMarker Client Control Properties

Use the following Properties settings to control how your OCX client works with the MapMarker Server and how it identifies candidates for geocoding.

Connections Property Page
On the Connections tab identify the name of the machine on which MapMarker Server is running. If you do not specify the server machine name, it assumes the server is running locally and looks for it there.

If you do not wish to start a service, for example, you are using a standalone one-user, one-machine system, select the DO NOT USE THE MAPMARKER SERVER check box. In that case, the OCX directly accesses the MapMarker geocoding engine. (Be sure that the MapMarker Control mapmarkr.ocx is located in the same directory as the geoengine mm32v10.dll.)

Match Restrictions Property Page
Control how MapMarker matches an address by setting the conditions here. Choose from require exact match on house number, street number and ZIP Code; match to street intersections; expand the search area, and list all candidates or only those that MapMarker determines are close matches.

Offset Property Page
Defines the distance from the corner and the street centerline the geocoded point spots.

General Property Page
You can specify an address to geocode at design time by supplying it in this property page.

About MM OCX Page
This page identifies the version of the OCX and provides MapInfo Corporation contact information.
Geocoding via MapMarker Geocoder Control

When the executable is run, a request is sent to MapMarker Server to geocode an address. The Server returns zero, one, or more match candidates. Each candidate contains the firm, address, match precision (e.g., street level or ZIP+4 centroid match), latitude/longitude coordinates, and Census Block ID.) The status line in the interface explains the type of match:

<table>
<thead>
<tr>
<th>If the Server returns...</th>
<th>the Status line displays...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINGLE_MATCH</td>
<td>Single close match</td>
</tr>
<tr>
<td>MULTIPLE_MATCHES</td>
<td>Multiple possible matches</td>
</tr>
<tr>
<td>NO_MATCHES</td>
<td>No close match candidates found -or- Candidates found*</td>
</tr>
<tr>
<td>UNIQUE_ZIP_NO_MATCH</td>
<td>No match to record containing unique ZIP Code (CASS mode only)</td>
</tr>
<tr>
<td>NO_CANDIDATES</td>
<td>No candidates found</td>
</tr>
<tr>
<td>POSSIBLE_INTERSECTION</td>
<td>Possible intersection</td>
</tr>
<tr>
<td>NO_INTERSECTION_MATCH</td>
<td>Intersection was not found</td>
</tr>
<tr>
<td>SINGLE_INTERSECT_MATCH</td>
<td>Single intersection match</td>
</tr>
<tr>
<td>MULTIPLE_INTERSECT_MATCH</td>
<td>Multiple possible intersection matches</td>
</tr>
<tr>
<td>GEO_ENG_BAD_INPUT_ADDRESS</td>
<td>Bad Input Address</td>
</tr>
</tbody>
</table>

* This message depends on the "List Candidates" setting under Match Restrictions.

Bad Input Address Error

A number of conditions generate Bad Input Address error messages in the status information field of the OCX. The following list describes the possible cases and indicates at which level they occur.

At OCX Level

- The address and ZIP Code fields are both empty.
- The city/state pair and ZIP Code are both empty.
- The city or state is misspelled or an invalid ZIP Code is supplied.

To avoid this error condition you should supply:

- A ZIP Code (only) or
- Street and ZIP Code or
- Street and city/state pair or
- Street, city/state pair and ZIP Code
At the GeoEngine Level

The Bad Input Address message may still be returned even if you supply a valid street address and a valid city/state combination but omit the ZIP Code. This occurs when a match on ZIP Code is required, but a ZIP Code has not been entered. Check the MapMarker Server and OCX geocoding parameters.

Falling Back to ZIP Code Centroids

MapMarker OCX falls back to a ZIP Code centroid match when there are no match candidates or the input address is bad. There are three situations in which MapMarker OCX falls back to ZIP Code Centroid:

- The operator entered a ZIP Code only.
- The operator entered a street and ZIP Code and MapMarker returns a Bad Input Address message.
- The operator entered a street and ZIP Code and MapMarker returned a message of No Candidates.

MapMarker Geocoder Control Events and Methods

When you use the supplied MapMarker Geocoder Control interface, certain events and methods from the OLE Automation API are called. They are included here as a summary. The full discussion of the OLE Automation API is found in Chapter 10: OLE Automation API.

<table>
<thead>
<tr>
<th>Event</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LatLongChanged()</td>
<td>This event is triggered when the Geocode button is pressed, the DoGeocode() method is called, or when a different candidate is selected in the Match Candidates list box. This event updates the latitude, longitude and precision properties.</td>
</tr>
<tr>
<td>GeocodeEvent()</td>
<td>This event is triggered when the Geocode button is pressed and the whole geocoding process is completed. Note that the DoGeocode() method does not trigger this event.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearDialogText()</td>
<td>Clears all visible text in the dialog, as well as these properties: firm, street, city, state, Zip, ZipPlus4, lastErrorCode, longitude, latitude, and precision.</td>
</tr>
<tr>
<td>RefreshDialog()</td>
<td>Repaints the dialog box. MapMarker Client Control usually repaints automatically and you do not need to call it.</td>
</tr>
<tr>
<td>Method</td>
<td>Purpose</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DoGeocode()</td>
<td>This method is the same as pressing the Geocode button. It implies that you must initialize these properties: firm, street, city, state, Zip, ZipPlus4, and serverName.</td>
</tr>
<tr>
<td>GetCandidateAt()</td>
<td>Returns the line of text in the Match Candidate list box.</td>
</tr>
<tr>
<td>GetCandidateFirmAt()</td>
<td>Returns the firm portion of the address as a string.</td>
</tr>
<tr>
<td>GetCandidateStreetAt()</td>
<td>Returns the street portion of the address as a string.</td>
</tr>
<tr>
<td>GetCandidateCityAt()</td>
<td>Returns the city portion of the address as a string.</td>
</tr>
<tr>
<td>GetCandidateStateAt()</td>
<td>Returns the state portion of the address as a string.</td>
</tr>
<tr>
<td>GetCandidateZIPAt()</td>
<td>Returns the ZIP Code portion of the address as a string.</td>
</tr>
<tr>
<td>GetCandidatePlus4At()</td>
<td>Returns the ZIP Code add on of the address as a string.</td>
</tr>
<tr>
<td>GetCandidatePrecisionAt()</td>
<td>Returns the precision for the match as a string. Precision refers to the quality of the match: to street level, shape path, intersection, Point ZIP, or ZIP centroid.</td>
</tr>
<tr>
<td>GetCandidateLongitudeAt()</td>
<td>Returns the longitude as a double.</td>
</tr>
<tr>
<td>GetCandidateLatitudeAt()</td>
<td>Returns the latitude as a double.</td>
</tr>
<tr>
<td>GetCandidateCensusBlockIDAt()</td>
<td>Returns the Census Block ID as a string.</td>
</tr>
<tr>
<td>DoSetProperties()</td>
<td>Brings up a set of property pages for the Client Control.</td>
</tr>
<tr>
<td>SelectCandidateAt()</td>
<td>Highlights the specified candidate in the Match Candidate List dialog box.</td>
</tr>
<tr>
<td>GetName()</td>
<td>Returns the application name.</td>
</tr>
<tr>
<td>GetFullName()</td>
<td>Returns the full name and path of the application.</td>
</tr>
<tr>
<td>GetVersionNum()</td>
<td>Returns the version number of the OCX.</td>
</tr>
<tr>
<td>GeocodeCheckDbAvailability()</td>
<td>Returns a number representing the type of database(s) available for geocoding.</td>
</tr>
<tr>
<td>GeocodeGetServerVersion()</td>
<td>Returns the server version number.</td>
</tr>
<tr>
<td>GeocodeGetStatesLicensed()</td>
<td>Returns a list of licensed states.</td>
</tr>
<tr>
<td>GeocodeGetStatesFound()</td>
<td>Returns a list of states found in the Address Dictionary path.</td>
</tr>
</tbody>
</table>
Client/Server Geocoding and the Internet

The MapMarker Client/Server toolkit is an excellent way to create geocoding applications via the Internet. The most common configuration for use of the toolkit and a web server is to create an Active Server Page (ASP) or ColdFusion application that uses the MapMarker Geocoder Control. The MapMarker client then sends the input address to the MapMarker Server via RPC. As your web server receives geocoding requests it executes the program for each request. The MapMarker Server obtains the requests via the MapMarker Geocoder Control, queues them, and then serves each one in turn.

For details and sample applications using ColdFusion functions, see Chapter 11: OLE Automation API for ColdFusion. For details and sample applications using ASP/VBScript functions, see Chapter 12: OLE Automation API – For ASP/VBScript.

Another solution is to use the MapMarker geocoding engine by itself via a web server API such as the Netscape API or the Microsoft Internet Server API. This alternative uses less system resources when processing geocoding requests. The disadvantage is that it requires more programming as well as a good understanding of multi-threaded programming.
This chapter explains the OLE Automation API. Use this to build your own geocoding control for your client application instead of using the program-ready MapMarker Geocoder Control (OCX), described in Chapter 9: Client/Server Geocoding.

In this chapter:

- Creating a Custom Geocoding Control ........................................... 130
- Summary of Geocoder Control Properties, Events, and Methods ................. 131
- OLE Automation Methods .......................................................... 135
- OLE Automation Objects ............................................................ 171
- Programming Usage Example ...................................................... 174
Creating a Custom Geocoding Control

If you find that the ready-made MapMarker Geocoder Control does not give you the flexibility you need for your application, consider creating your own OLE Automation control that calls the MapMarker Server and carries out the geocoding operation. The MapMarker Geocoder Control is based on the same OLE Automation interface, so you have access to the same methods and properties in your custom control.

Note that neither the MapMarker geocoding engine nor the MapMarker RPC server is multi-threaded.

The following diagram illustrates the OLE Automation methods involved in the geocoding operation. This is the flow of events for the MapMarker Geocoder Control as well as the recommended flow if you are using the OLE Automation API without the Geocoder Control user interface.
Summary of Geocoder Control Properties, Events, and Methods

The following tables define the properties, events, and methods that come into play when developing a custom geocoding application using the MapMarker Geocoder Control and/or the OLE Automation API.

### Input Properties

<table>
<thead>
<tr>
<th>Input Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firm</td>
<td>String: Firm name</td>
</tr>
<tr>
<td>Street</td>
<td>String: Street Address</td>
</tr>
<tr>
<td>Street2</td>
<td>String: Secondary Address</td>
</tr>
<tr>
<td>City</td>
<td>String: City</td>
</tr>
<tr>
<td>State</td>
<td>String: State</td>
</tr>
<tr>
<td>Zip</td>
<td>String: ZIP Code</td>
</tr>
<tr>
<td>ZipPlus4</td>
<td>String: ZIP Code add-on</td>
</tr>
<tr>
<td>ServerName</td>
<td>String: Name of Server where MapMarker Server is running</td>
</tr>
<tr>
<td>NotUsingServer</td>
<td>Boolean: If TRUE does not connect to the server. Place mapmarkr.ocx in the same directory as the geoengine mm32v_.dll. Default: False.</td>
</tr>
<tr>
<td>BCloseCandidates</td>
<td>Boolean: If TRUE, shows close matches only; if FALSE shows all possible matches. Set this property on the Match Restrictions tab of the Property Pages dialog. Default: False.</td>
</tr>
<tr>
<td>ExactHouse</td>
<td>Boolean: If TRUE, exact match on house number is required Default: True.</td>
</tr>
<tr>
<td>ExactName</td>
<td>Boolean: If TRUE, exact match on street name is required Default: False.</td>
</tr>
<tr>
<td>ExactZIP</td>
<td>Boolean: If TRUE, exact match on ZIP Code is required Default: False.</td>
</tr>
<tr>
<td>ExactCity</td>
<td>Boolean: If TRUE, exact match on City is required. Default: False.</td>
</tr>
<tr>
<td>ExpandSearch</td>
<td>Boolean: If TRUE, MapMarker expands the search area in which it looks for match candidates. Default: False.</td>
</tr>
<tr>
<td>ExpandSearchInState</td>
<td>Boolean: If TRUE, the expanded search area is limited to the state. Default: False.</td>
</tr>
<tr>
<td>ExpandDistance</td>
<td>Integer: If ExpandSearch is set to TRUE, this specifies the radius of the search in miles from the finance area centroid. Default: 0. A Finance Area centroid is an area defined by the U.S. Postal Service to collect cost and statistical data. A Finance Area is frequently used for search areas because it covers some or all the ZIP Code areas in a town or city.</td>
</tr>
</tbody>
</table>
## Input Properties

<table>
<thead>
<tr>
<th>Input Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchIntersections</td>
<td>Boolean: If TRUE, street intersection matching is attempted. Default: False.</td>
</tr>
<tr>
<td>PreferUserDictionary</td>
<td>Boolean: If TRUE, and the preferred user dictionary and Address dictionary are used together, the preferred user dictionary match is weighted higher than a match with the same score in the Address Dictionary. Default: False.</td>
</tr>
<tr>
<td>LinearOffset</td>
<td>Double: Defines the position of the geocoded point with respect to the corner. Default: 25.</td>
</tr>
<tr>
<td>PerpendicularSetback</td>
<td>Double: Defines the position of the geocoded point with respect to the centerline of the street. Default: 20.</td>
</tr>
<tr>
<td>Units</td>
<td>Integer: The units used in LinearOffset and PerpendicularSetback. Default: 0. These units are: INCH 2, LINK 3, SURVEY_FOOT 4, YARD 5, ROD 6, CHAIN 7, MILE 8, NAUTICAL_MILE 9, MILLIMETER 10, CENTIMETER 11, METER 12, KILOMETER 13</td>
</tr>
<tr>
<td>ShowPropertiesButton</td>
<td>Boolean: If TRUE, the Show Properties button is visible. Default: True.</td>
</tr>
<tr>
<td>ShowBorder</td>
<td>Boolean: If TRUE, the border around the control's interface is visible. Default: True.</td>
</tr>
<tr>
<td>CassMode</td>
<td>Boolean: If TRUE, CASSMode is set to on. Default: False</td>
</tr>
</tbody>
</table>

## Output Properties

<table>
<thead>
<tr>
<th>Output Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitude</td>
<td>Double: The latitude value for a match candidate.</td>
</tr>
<tr>
<td>Longitude</td>
<td>Double: the longitude value for a match candidate.</td>
</tr>
</tbody>
</table>
### Output Properties

<table>
<thead>
<tr>
<th>Output Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Precision | Integer: A number that identifies match precision (street level, shape path, intersection, point ZIP, or ZIP centroid).  
No Centroid 0  
ZIP Point Centroid 1  
ZIP+2 Centroid 2  
ZIP+4 Centroid 3  
Shape Path Center 10  
Street Address 20  
Street Intersection 30  
Point Zip 40 |
| NumCandidates | Integer: The number of match candidates for the record. |
| NumCloseCandidates | Integer: The number of close match candidates for the record. |
| LastErrorCode | Long: Returns the last error code generated. See Appendix H: MapMarker GeoEngine Error Codes, for descriptions of the error codes. |
| CensusBlockID | String: Census Block tabulation number. |
| ResultCode | String: Geocoding result code. |
| DatabaseTypes | Integer: a number that identifies the available databases:  
2 - ZIP Centroid database  
3 - Street and ZIP Code databases  
4 - User Dictionary  
6 - User Dictionary and ZIP Code databases  
7 - Street, User Dictionary, and ZIP Code databases |
| StringBinding | String: RPC binding string used to connect to the server. |
| PrimaryStreet | String: The candidate primary street, if it has one. |
| RecordType | String: The candidate ZIP+4 record type as categorized by the U.S. Postal Service. If the match is from a TIGER record, the candidate may not have a record type.  
F – Firm  
G – General Delivery  
H – Highrise  
P – PO Box  
R – Rural Route/Highway  
S – Street |
| DeliveryPoint | String: The candidate delivery point. |
| Carrier Route | String: The candidate carrier route. |
| CheckDigit | String: The candidate check digit. |
### Output Properties

<table>
<thead>
<tr>
<th>Output Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TabbedAddress</td>
<td>String: The candidate street address components delimited by tabs as follows: House Number, Directional Prefix, Street Type Prefix, Street Name, Street Type Suffix, Unit Type, Unit Value. Example: 271 E Deering Ave Apt 3.</td>
</tr>
<tr>
<td>Lacs</td>
<td>String: Single character output code that indicates that the record can be converted from a rural route address to a city-style address by using a USPS product, known as the Locatable Address Conversion Service.</td>
</tr>
<tr>
<td>Pmb</td>
<td>String: “Private Mailbox” used at Commercial Mail Receiving Agencies (CMRA) such as Mailboxes Etc.</td>
</tr>
<tr>
<td>PmbRange</td>
<td>String: The candidate Pmb range/number when there is a PMB range in the input.</td>
</tr>
<tr>
<td>AddressType</td>
<td>Integer. MapMarker address type. Street 10, Place 11, Zip 12, Rural 13, Highway 14, PO Box 15, Military 16, Intersection 17</td>
</tr>
</tbody>
</table>

### Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LatLongChanged()</td>
<td>This event is triggered when the Geocode button is pressed, the DoGeocode() method is called, or when a different candidate is selected in the Match Candidates list box. This event updates the latitude, longitude, and precision properties.</td>
</tr>
<tr>
<td>GeocodeEvent()</td>
<td>This event is triggered when the Geocode button is pressed and the whole geocoding process is completed. <strong>Note:</strong> The DoGeocode() method does not trigger this event.</td>
</tr>
</tbody>
</table>
OLE Automation Methods

The following pages contain the OLE Automation method descriptions in alphabetical order.

- **ClearDialogText()** 137
  Clears all visible text in the dialog, as well as these properties: firm, street, city, state, Zip, ZipPlus4, lastErrorCode, longitude, latitude, precision, resultCode, numCandidates, and numCloseCandidates.

- **Connect()** 137
  Builds an RPC binding string to connect to the MapMarker Server. Calls GeocodeCheckDbAvailability().

- **Disconnect()** 138
  Disconnects from the MapMarker Server.

- **DoGeocode()** 139
  This method is the same as pressing the Geocode button. It implies that you must initialize these properties: firm, street, city, state, Zip, ZipPlus4, serverName.

- **DoSetProperties()** 139
  Brings up a set of property pages for the Geocoder Control.

- **DpvGeocodeAddress()** 139
  Turns DPV mode on, and then geocodes an address and builds a list of candidates.

- **DpvGeocodeAddressLastLine()** 141
  Turns DPV mode on, and then allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database).

- **DpvGeocodeAddressWithSerial()** 143
  Turns DPV mode on, and then allows you to geocode an address when the server requires that a serial number be provided to allow geocoding.

- **GeocodeAddress()** 145
  Attempts to geocode an address and build a list of candidates.

- **GeocodeAddressEx()** 147
  Attempts to geocode an address and build a list of candidates. Also enables street2 information to be passed in as a separate field.

- **GeocodeAddressLastLine()** 148
  Enables you to enter an input address with an unparsed line containing the city, state, and/or ZIP Code.

- **GeocodeAddressLastLineEx()** 150
  Enables you to enter an input address with an unparsed last line containing the city, state, and/or ZIP code. Also enables you to pass in street2 information as a separate field.

- **GeocodeAddressLastLineWithSerial()** 151
  Enables you to geocode an address using an unparsed last line containing the city, state, and/or ZIP Code. Requires a user to input a serial number for geocoding.

- **GeocodeAddressWithSerial()** 152
  Requires user to input a serial number for geocoding.

- **GeocodeAddressWithSerialEx()** 153
  Requires user to input a serial number for geocoding, and enables street2 information to be passed in as a separate field.

- **GeocodeCheckDbAvailability()** 155
  Checks to see which types of database(s) are available for geocoding.
- **GeocodeFreeSet()** ................................................. 155
  Frees the server side information after geocoding.
- **GeocodeGetCandidates()** ...................................... 156
  Gets all the candidate information after geocoding.
- **GeocodeGetErrorText()** ........................................ 157
  Returns a short text description of the specified error.
- **GeocodeGetHwyExitCandidate()** ......................... 158
  Gets the specified highway exit candidate after a call to GeocodeHwyExit_A().
- **GeocodeGetServerVersion()** .............................. 159
  Returns the server version number.
- **GeocodeGetStatesFound()** .................................. 159
  Returns a list of 2-digit state abbreviations representing states found in the Address Dictionary path. These are not necessarily licensed states.
- **GeocodeGetStatesLicensed()** ............................... 160
  Returns a list of 2-digit state abbreviations representing licensed states.
- **GeocodeHwyExit()** ........................................... 160
  Returns a count of the number of highway records based on input criteria of highway exit and state.
- **GeocodesCandidateMultiUnit()** ..................... 161
  Returns a True/False value, indicating whether the candidate address contains multiple units.
- **GeocodePostalCentroid()** .................................. 162
  Geocodes a ZIP Code or ZIP+4 centroid.
- **GeocodePostalCentroidWithSerial()** ...... 163
  Requires a serial number for ZIP Code-level geocoding.
- **GetCandidateAt()** ........................................... 164
  Returns the line of text in the Match Candidate list box.
- **GetCandidateCensusBlockIDAt()** .................. 164
  Returns the Census Block ID as a string.
- **GetCandidateCityAt()** ...................................... 165
  Returns the city portion of the address as a string.
- **GetCandidateFirmAt()** ...................................... 165
  Returns the firm portion of the address as a string.
- **GetCandidateLatitudeAt()** ............................... 166
  Returns the latitude as a double.
- **GetCandidateLongitudeAt()** .............................. 166
  Returns the longitude as a double.
- **GetCandidatePlus4At()** ..................................... 167
  Returns the ZIP add on of the address as a string.
- **GetCandidatePrecisionAt()** .............................. 167
  Returns the precision for the match as a string. Precision refers to the quality of the match: to street level, shape path, intersection, point ZIP, or ZIP centroid.
- **GetCandidateResultCodeAt()** ....................... 168
  Returns the result code portion of the address as a string.
- **GetCandidateStateAt()** ...................................... 168
  Returns the state portion of the address as a string.
- **GetCandidateStreetAt()** .................................... 169
  Returns the street portion of the address as a string.
- **GetCandidateZIPAt()**  .......................................................... 169
  Returns the ZIP portion of the address as a string.
- **GetFullName()**  ................................................................. 170
  Returns the full name and path of the application.
- **GetName()**  ....................................................................... 170
  Returns the application name.
- **GetVersionNum()** ................................................................. 170
  Returns the version number of the application.
- **RefreshDialog()** ................................................................. 170
  Forces the dialog box to repaint.
- **SelectCandidateAt()** ............................................................ 171
  Highlights the specified candidate in the Match Candidate List box.

### ClearDialogText()

**Purpose**

This call clears all visible text in the MapMarker Geocoder Control’s interface. It also clears the Firm, Street, City, State, Zip, ZipPlus4, LastErrorCode, Latitude, Longitude, Precision, NumCandidates, ResultCode, numCloseCandidates, CensusBlockID, and StringBinding properties. It also generates the LatLongChanged() event.

**Syntax**

```plaintext
ClearDialogText()
```

**Returns**

The LastError Code property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

### Connect()

**Purpose**

This call builds an RPC binding string to connect to the MapMarker Server and checks the type(s) of databases that are available for geocoding.

It attempts to connect to the MapMarker RPC server. If the server is NULL or is a local server, it attempts to establish a connection using LRPC. Otherwise it uses TCP/IP and dynamic endpoint binding to attempt a connection.

**Syntax**

```plaintext
Connect(NetworkAddress As String) As Boolean
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetworkAddress</td>
<td>[String] identifies the name or IP Address of the machine on which MapMarker Server is running. NULL or an empty string assumes MapMarker Server is running locally, and attempts to use local RPC.</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>10</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
<tr>
<td>GEO_ENG_MISSING_DATABASE_ERR</td>
<td>1078</td>
<td>Address Dictionary file(s) not found.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_NOT_FOUND_ERR</td>
<td>1228</td>
<td>Zipmastr.cdb or Zipmastr.jdx not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_LICFILE</td>
<td>1079</td>
<td>License file not found.</td>
</tr>
<tr>
<td>GEO_ENG_INVALID_SERIAL_NUMBER</td>
<td>1085</td>
<td>Invalid serial number.</td>
</tr>
<tr>
<td>GEO_ENG_PARSE_INIT_ERR</td>
<td>307</td>
<td>Parsing file (geo_usa.*) missing or bad.</td>
</tr>
<tr>
<td>GEO_ENG_MATCH_INIT_ERR</td>
<td>563</td>
<td>Matching file (geo_usa.*) missing or bad.</td>
</tr>
<tr>
<td>GEO_ENG_NADCON_MISSING_FILE</td>
<td>1334</td>
<td><em>.las/</em>.los files not found.</td>
</tr>
</tbody>
</table>

Disconnect()

Purpose

This call is used to disconnect from the MapMarker Server.

Syntax

```
Disconnected() As Boolean
```

Returns

TRUE if successful, FALSE if not connected or if an error occurs. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
**DoGeocode()**

**Purpose**
This call does the same thing as clicking the Geocode button. Using this call assumes that the Firm, Street, City, State, Zip, ZipPlus4, and ServerName properties are initialized. Updates the longitude, latitude, precision, Census Block ID, result code, NumCandidates, NumCloseCandidates, etc.

**Syntax**
```
DoGeocode()
```

**DoSetProperties()**

**Purpose**
This call brings up a set of property pages for the MapMarker Geocoder Control.

**Syntax**
```
DoSetProperties() As Boolean
```

The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

**DpvGeocodeAddress()**

**Purpose**
This call turns DPV mode on, and then geocodes an address and builds a list of candidates. It also enables you to pass street2 information as a separate field. Note that the CASSMode property must be set to True or the method will be unable to enable DPV mode.

**Note:** This method is not for use with VBScript or ColdFusion. The appropriate call for VB is DpvGeocodeAddressV() (page 204). A corresponding call is not available for ColdFusion.

**Syntax**
```
DpvGeocodeAddress(geocodeHandle As Long, 
  firm As String, 
  street As String, 
  street2 As String, 
  city As String, 
  state As String, 
  zip As String, 
  status As Integer, 
  numCandidates As Integer, 
  numCloseCandidates As Integer) As Boolean
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
<tr>
<td>status</td>
<td>[Output Integer] One of the following: -1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code (CASS mode only). 0 – SINGLE_MATCH – a single close match was found for the input address. 1 – MULTIPLE_MATCH – multiple match candidates were found and MapMarker could not choose between at least two of them. 2 – NO_MATCHES – candidates found, but none considered a close match. 3 – NO_CANDIDATES – no candidates found for the input address. 4 – SINGLE_INTERSECT_MATCH – an intersection match was found. 5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was found. 6 – NO_INTERSECT_MATCHES – no close intersection candidates found. 7 – NO_INTERSECT_CANDIDATES – no intersection candidates found. 8 – POSSIBLE_INTERSECTION – possible close intersection match found.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output Integer] Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] Number of close candidates.</td>
</tr>
</tbody>
</table>

Returns

TRUE if successful, FALSE if not. The LastError Code property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
DpvGeocodeAddressLastLine()

Purpose
This method turns DPV mode on, and then allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database). This method also enables you to pass in street2 information as a separate field. Note that the CASSMode property must be set to True or the method will be unable to enable DPV mode.

Note: This method is not for use with VBScript or ColdFusion. The appropriate call for VB is DpvGeocodeAddressLastLineV() (page 205). A corresponding call is not available for ColdFusion.

Syntax
DpvGeocodeAddressLastLine(geocodeHandle As Long, firm As String, street As String, street2 As String, lastline As String, status As Integer, numCandidates As Integer, numCloseCandidates As Integer) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>lastline</td>
<td>[Input String] Unparsed last line of address to be matched.</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

DpvGeocodeAddressLastLineWithSerial()

Purpose

This method turns DPV mode on, and then allows you to geocode an address using an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database) instead of separate city, state, and ZIP fields. Use this call in server applications that are initialized with a serial number. Note that the CASSMode property must be set to True or the method will be unable to enable DPV mode. This method should now be used instead of GeocodeAddressLastLine, which used to have an optional serial number parameter.

Note: This method is not for use with VBScript or ColdFusion. The appropriate call for VB is DPVGeocodeAddressLastLineV() (page 205). A corresponding call for ColdFusion is not available.
Syntax

```vba
DpvGeocodeAddressLastlineWithSerial(geocodeHandle as Long,
   Firm as String,
   Street as String,
   Street2 as String,
   Lastline as String,
   Status as Integer,
   NumCandidates as Integer,
   NumCloseCandidates as Integer,
   SerialNumber as String)As Boolean
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>Firm</td>
<td>[Input String] Firm name.</td>
</tr>
<tr>
<td>Street</td>
<td>[Input String] Street name.</td>
</tr>
<tr>
<td>Street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>Lastline</td>
<td>[Input String] Lastline information (city, state, ZIP).</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output Integer] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close match candidates.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] The serial number for OEM license usage.</td>
</tr>
</tbody>
</table>

Returns

The call returns TRUE if successful; FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

DpvGeocodeAddressWithSerial()

Purpose

This method turns DPV mode on, and then allows you to geocode an address when the server requires that a serial number be provided to allow geocoding. Note that the CASSMode property must be set to True or the method will be unable to enable DPV mode.

This method is not for use with VBScript or ColdFusion. The appropriate call for VB is GeocodeAddressV() (page 204). A corresponding call for ColdFusion is not available.
Syntax

DpvGeocodeAddressWithSerial(geocodeHandle As Long,
    firm As String,
    street As String,
    street2 As String,
    city As String,
    state As String,
    zip As String,
    status As Integer,
    numCandidates As Integer,
    numCloseCandidates As Integer,
    SerialNumber As String) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
<tr>
<td>status</td>
<td>[Output Integer] One of the following:</td>
</tr>
<tr>
<td></td>
<td>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code (CASS mode only).</td>
</tr>
<tr>
<td></td>
<td>0 – SINGLE_MATCH – a single close match was found for the input address.</td>
</tr>
<tr>
<td></td>
<td>1 – MULTIPLE_MATCH – multiple match candidates were found and Map-Marker could not choose between at least two of them.</td>
</tr>
<tr>
<td></td>
<td>2 – NO_MATCHES – candidates found, but none considered a close match.</td>
</tr>
<tr>
<td></td>
<td>3 – NO_CANDIDATES – no candidates found for the input address.</td>
</tr>
<tr>
<td></td>
<td>4 – SINGLE_INTERSECT_MATCH – an intersection match was found.</td>
</tr>
<tr>
<td></td>
<td>5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was found.</td>
</tr>
<tr>
<td></td>
<td>6 – NO_INTERSECT_MATCHES – no close intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>7 – NO_INTERSECT_CANDIDATES – no intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>8 – POSSIBLE_INTERSECTION – possible close intersection match found.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output Integer] The total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close candidates.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] Serial number required by server, created by an OEM license program sold separately.</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GeocodeAddress()

Purpose

This is the main call for the OLE Automation API. It attempts to geocode an address and build a list of candidates.

Note: This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodeAddressV() (page 207) and GeocodeAddress_A() (page 187), respectively.

Syntax

GeocodeAddress(geocodeHandle As Long, firm As String, street As String, city As String, state As String, zip As String, status As Integer, numCandidates As Integer, numCloseCandidates As Integer) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

Usage Example

The following is a simplified version of a VB function for geocoding. User interface handling, error handling, and other details of coding are removed in order to show the processing flow more clearly.

See Programming Usage Example on page 174, for a complete sample program that illustrates this function.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>[Output Integer] One of the following:</td>
</tr>
<tr>
<td></td>
<td>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code (CASS mode only).</td>
</tr>
<tr>
<td></td>
<td>0 – SINGLE_MATCH – a single close match was found for the input address.</td>
</tr>
<tr>
<td></td>
<td>1 – MULTIPLE_MATCH – multiple match candidates were found and MapMarker could not choose between at least two of them.</td>
</tr>
<tr>
<td></td>
<td>2 – NO_MATCHES – candidates found, but none considered a close match.</td>
</tr>
<tr>
<td></td>
<td>3 – NO_CANDIDATES – no candidates found for the input address.</td>
</tr>
<tr>
<td></td>
<td>4 – SINGLE_INTERSECT_MATCH – an intersection match was found.</td>
</tr>
<tr>
<td></td>
<td>5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was found.</td>
</tr>
<tr>
<td></td>
<td>6 – NO_INTERSECT_MATCHES – no close intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>7 – NO_INTERSECT_CANDIDATES – no intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>8 – POSSIBLE_INTERSECTION – possible close intersection match found.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output Integer] Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] Number of close candidates.</td>
</tr>
</tbody>
</table>
Private Sub cmdGeocode_Click()
Dim retVal As Boolean
Dim retCode
retVal = objMM.GeocodeAddress(lngEngHandle, txtfirm, txtstreet, 
txtcity, txtstate, txtzip, status, numCandidates, 
numCloseCandidates)
If Not retVal Then
    'Error handling
    '... ...
Else
    'Show geocoding status
    Select Case status
    Case 0
        lblStatus = "Single match"
    Case 1
        lblStatus = "Multiple match"
    Case 2
        lblStatus = "No close matches"
    Case 3
        lblStatus = "No candidates"
    Case 4
        lblStatus = "Single intersection match"
    Case 5
        lblStatus = "Multiple intersection match"
    Case 6
        lblStatus = "No close intersection matches"
    Case 7
        lblStatus = "No intersection candidates"
    Case 8
        lblStatus = "Possible intersections"
    Case Else
        lblStatus = Str$(status)
    End Select
End If
End Sub

GeocodeAddressEx()

Purpose
This call attempts to geocode an address and build a list of candidates. It works similarly to
GeocodeAddress(); however, it also enables you to pass in street2 information as a separate field.

Note: This method is not for use with VBScript or ColdFusion. The appropriate calls are
GeocodeAddressExV() (page 208) and GeocodeAddressEx_A() (page 188), respectively.

Syntax
GeocodeAddressEx(geocodeHandle as long, 
    Firm as String, 
    Street as String, 
    Street2 as String, 
    City as String, 
    State as String, 
    Zip as String, 
    Status as Integer, 
    NumCandidates as Integer, 
    NumCloseCandidates as Integer)As Boolean
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>Firm</td>
<td>[Input String] Firm name.</td>
</tr>
<tr>
<td>Street</td>
<td>[Input String] Street name.</td>
</tr>
<tr>
<td>Street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>City</td>
<td>[Input String] City.</td>
</tr>
<tr>
<td>State</td>
<td>[Input String] State.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output Integer] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close match candidates.</td>
</tr>
</tbody>
</table>

Returns

The call returns TRUE if successful; FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GeocodeAddressLastLine()

Purpose

This method allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database).

Note: This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodeAddressV() (page 207) and GeocodeAddress_A() (page 187), respectively.

Syntax

```vba
GeocodeAddressLastLine(geocodeHandle As Long,
    firm As String,
    street As String,
    lastline As String,
    status As Integer,
    numCandidates As Integer,
    numCloseCandidates As Integer) As Boolean
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>lastline</td>
<td>[Input String] Unparsed last line of address to be matched.</td>
</tr>
<tr>
<td>status</td>
<td>[Output Integer] One of the following:</td>
</tr>
<tr>
<td></td>
<td>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code (CASS mode only).</td>
</tr>
<tr>
<td></td>
<td>0 – SINGLEMATCH – a single close match was found for the input address.</td>
</tr>
<tr>
<td></td>
<td>1 – MULTIPLEMATCH – multiple match candidates were found and MapMarker could not choose between at least two of them.</td>
</tr>
<tr>
<td></td>
<td>2 – NO_MATCHES – candidates found, but none considered a close match.</td>
</tr>
<tr>
<td></td>
<td>3 – NO_CANDIDATES – no candidates found for the input address.</td>
</tr>
<tr>
<td></td>
<td>4 – SINGLE_INTERSECT_MATCH – an intersection match was found.</td>
</tr>
<tr>
<td></td>
<td>5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was found.</td>
</tr>
<tr>
<td></td>
<td>6 – NO_INTERSECT_MATCHES – no close intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>7 – NO_INTERSECT_CANDIDATES – no intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>8 – POSSIBLE_INTERSECTION – possible close intersection match found.</td>
</tr>
</tbody>
</table>

| numCandidates   | [Output Integer] The total number of candidates found.                                                                                                                                        |
| numCloseCandidates | [Output Integer] The number of close candidates.                                                                                                                                          |

Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GeocodeAddressLastlineEx()

Purpose

This method allows you to geocode an address using an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database) instead of separate city, state, and ZIP fields. It works similarly to GeocodeAddressLastline(); however, it also enables you to pass in street2 information as a separate field.

Note: This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodeAddressLastlineExV() (page 211) and GeocodeAddressLastlineEx_A() (page 150), respectively.

Syntax

```vbnet
GeocodeAddressLastlineEx(geocodeHandle as Long,
Firm as String,
Street as String,
Street2 as String,
lastline as String,
status as Integer,
NumCandidates as Integer,
NumCloseCandidates as Integer)As Boolean
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>Firm</td>
<td>[Input String] Firm name.</td>
</tr>
<tr>
<td>Street</td>
<td>[Input String] Street name.</td>
</tr>
<tr>
<td>Street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>Lastline</td>
<td>[Input String] Lastline information (city, state, ZIP)</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output Integer] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close match candidates.</td>
</tr>
</tbody>
</table>

Returns

The call returns TRUE if successful; FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GeocodeAddressLastLineWithSerial()

Purpose
This method allows you to geocode an address using an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database) instead of separate city, state, and ZIP fields. Use this call in server applications that are initialized with a serial number. This method should now be used instead of GeocodeAddressLastline, which used to have an optional serial number parameter.

Note: This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodeAddressLastLineV() (page 210) and GeocodeAddressLastLine_A() (page 189), respectively.

Syntax
GeocodeAddressLastLineWithSerial(geocodeHandle as Long, Firm as String, Street as String, Street2 as String, Lastline as String, Status as Integer, NumCandidates as Integer, NumCloseCandidates as Integer, SerialNumber as String)As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>Firm</td>
<td>[Input String] Firm name.</td>
</tr>
<tr>
<td>Street</td>
<td>[Input String] Street name.</td>
</tr>
<tr>
<td>Street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>Lastline</td>
<td>[Input String] Lastline information (city, state, ZIP).</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output Integer] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close match candidates.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] The serial number for OEM license usage.</td>
</tr>
</tbody>
</table>

Returns
The call returns TRUE if successful; FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GeocodeAddressWithSerial()

Purpose

This routine is called to geocode an address when the server requires that a serial number be provided to allow geocoding.

This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodeAddressV() (page 207) and GeocodeAddress_A() (page 187), respectively.

Syntax

```
GeocodeAddressWithSerial(geocodeHandle As Long,
    firm As String,
    street As String,
    city As String,
    state As String,
    zip As String,
    status As Integer,
    numCandidates As Integer,
    numCloseCandidates As Integer,
    SerialNumber As String) As Boolean
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
</tbody>
</table>
GeocodeAddressWithSerialEx()

Purpose
This routine is called to geocode an address when the server requires that a serial number be provided to allow geocoding. This method is for OEM users only and it requires an additional program to create the OEM licenses (purchased separately from MapInfo Corporation).

This call works similarly to GeocodeAddressWithSerial(); however, it also enables you to pass in street2 information in a separate field.

Returns
TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>[Output Integer] One of the following:</td>
</tr>
<tr>
<td></td>
<td>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code</td>
</tr>
<tr>
<td></td>
<td>(CASS mode only).</td>
</tr>
<tr>
<td></td>
<td>0 – SINGLE_MATCH – a single close match was found for the input address.</td>
</tr>
<tr>
<td></td>
<td>1 – MULTIPLE_MATCH – multiple match candidates were found and MapMarker</td>
</tr>
<tr>
<td></td>
<td>could not choose between at least two of them.</td>
</tr>
<tr>
<td></td>
<td>2 – NO_MATCHES – candidates found, but none considered a close match.</td>
</tr>
<tr>
<td></td>
<td>3 – NO_CANDIDATES – no candidates found for the input address.</td>
</tr>
<tr>
<td></td>
<td>4 – SINGLE_INTERSECT_MATCH – an intersection match was found.</td>
</tr>
<tr>
<td></td>
<td>5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was found.</td>
</tr>
<tr>
<td></td>
<td>6 – NO_INTERSECT_MATCHES – no close intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>7 – NO_INTERSECT_CANDIDATES – no intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>8 – POSSIBLE_INTERSECTION – possible close intersection match found.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output Integer] The total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close candidates.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] Serial number required by server, created by an OEM license</td>
</tr>
<tr>
<td></td>
<td>program sold separately.</td>
</tr>
</tbody>
</table>
Syntax

    GeocodeAddressWithSerialEx(geocodeHandle as long,
        Firm as String,
        Street as String,
        Street2 as String,
        City as String,
        State as String,
        Zip as String,
        Status as Integer,
        NumCandidates as Integer,
        NumCloseCandidates as Integer,
        SerialNumber as String)As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>Firm</td>
<td>[Input String] Firm name.</td>
</tr>
<tr>
<td>Street</td>
<td>[Input String] Street name.</td>
</tr>
<tr>
<td>Street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>City</td>
<td>[Input String] City name.</td>
</tr>
<tr>
<td>State</td>
<td>[Input String] State name.</td>
</tr>
<tr>
<td>Zip</td>
<td>[Input String] ZIP Code (includes ZIP + 4)</td>
</tr>
<tr>
<td>NumCandidates</td>
<td>[Output Integer] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close match candidates.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] The serial number for OEM license usage.</td>
</tr>
</tbody>
</table>

Returns

This call returns TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
**GeocodeCheckDbAvailability()**

**Purpose**
This method verifies that a valid database is available. The DatabaseTypes property is set as follows depending on which databases are present:

- 2–ZIP Centroid database
- 3–Street and ZIP Code databases
- 4–User Dictionary
- 6–User Dictionary and ZIP Code databases
- 7–Street, User Dictionary, and ZIP Code databases

**Syntax**
```
GeocodeCheckDbAvailability() As Boolean
```

**Returns**
TRUE if a valid database is available, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

---

**GeocodeFreeSet()**

**Purpose**
Whenever GeocodeAddress() is invoked, a memory structure is allocated, the structure is populated with a list of possible candidates and a context handle to that structure is returned to the caller. GeocodeFreeSet() releases the memory structure allocated to hold the candidate list. Once the information is freed, the handle cannot be used again, unless the same value is returned by a later call to GeocodeAddress().

MapMarker can allocate a maximum of 1,024 of these structures. To avoid running out of memory or blocking the MapMarker Server, call GeocodeFreeSet() as soon as you have finished evaluating a candidate list.

**Note:** This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodeFreeSetV() (page 212) and GeocodeFreeSet_A() (page 193), respectively.

**Syntax**
```
GeocodeFreeSet(geocodeHandle As Long) As Boolean
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input/Output] Used to reference information about the geocoding operation in other calls.</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GeocodeGetCandidates()

Purpose

This call returns all the needed candidate information from the GeocodeAddress() call. This information is stored in a collection class.

Note: This method is not for use with VBScript. The appropriate call is GeocodeGetCandidatesV() (page 212).

Syntax

GeocodeGetCandidates(geocodeHandle As Long, numCandidates As Integer) As Object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input Long] Used to reference information about the most recent geocoding operation.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Input Integer] Specifies the number of candidates that should be returned (e.g., the size of the collection).</td>
</tr>
</tbody>
</table>

Returns

A CAddressList collection of CAddress objects. The number of addresses to be stored in the collection is the input candidate number. This number must be less than or equal to the total number of candidates. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
Usage Example
The following is a simplified version of a VB function for getting candidate addresses. User interface handling, error handling and other details of coding are removed in order to show the processing flow more clearly. See Programming Usage Example on page 174, for a complete sample program that illustrates this function.

```vbnet
Private Sub cmdGetCand_Click()
    Dim adrList As CAddressList
    Dim adr As CAddress
    Dim strAddr$, strCoords$, strPrec$, strCensId$
    Set adrList = objMM.GeocodeGetCandidates(lngEngHandle,
                                              numCandidates)
    'Fill in the ListBox with the candidates
    lstCandidates.Clear
    For Each adr In adrList
        With adr
            'Set strAddr from .Street and other fields of adr
            'Set strCoords from .Latitude and .Longitude of adr
            'Set strPrec from .Precision of adr
            'Set strCensId from .CensusBlock of adr
            End With
            Set adr = Nothing
        'Add this candidate to the listbox
        lstCandidates.AddItem (strAddr & strCoords & strPrec &
                                    strCensId)
    Next
    Set adrList = Nothing
End Sub
```

GeocodeGetErrorText()

Purpose
Given a valid MapMarker error code this method returns a string giving a short text description of the error.

Note: This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodeGetErrorTextV() (page 213) and GeocodeGetErrorText_A() (page 194), respectively.

Syntax

`GeocodeGetErrorText(errorcode as Integer, errorString as String) as Boolean`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorString</td>
<td>[Output String] Error text.</td>
</tr>
</tbody>
</table>

Returns

The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GeocodeGetHwyExitCandidate()

Purpose
Given a zero-based index number, this method returns highway name, exit number, and/or exit name, as well as the ZIP Code, city, and coordinates of the exit location. This method cannot be used until its corresponding call, GeocodeHwyExit(), has been invoked. Using the count from that function, the individual candidates can be retrieved from GeocodeGetHwyExitCandidate(). The first candidate is 0.

Syntax

```
GeocodeGetHwyExitCandidate(
    index as Integer,
    HwyTypePre as String,
    HwyName as String,
    HwyTypeSuf as String,
    HwyDir as String,
    ExitNum as String,
    ExitName as String,
    City as String,
    State as String,
    Zipcode as String,
    Longitude as Double,
    Latitude as Double)As Boolean
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>[Input Integer] Zero-based index of candidate to retrieve.</td>
</tr>
<tr>
<td>HwyTypePre</td>
<td>[Output String] Type Prefix of the matched highway (e.g., Highway 21).</td>
</tr>
<tr>
<td>HwyName</td>
<td>[Output String] Name of the matched highway.</td>
</tr>
<tr>
<td>HwyTypeSuf</td>
<td>[Output String] Type Suffix of the matched highway (e.g., Pennsylvania Turnpike).</td>
</tr>
<tr>
<td>HwyDir</td>
<td>[Output String] The direction of the matched highway.</td>
</tr>
<tr>
<td>ExitNum</td>
<td>[Output String] The matched exit number.</td>
</tr>
<tr>
<td>ExitName</td>
<td>[Output String] The matched exit name.</td>
</tr>
<tr>
<td>City</td>
<td>[Output String] The returned city name.</td>
</tr>
<tr>
<td>State</td>
<td>[Output String] The returned state name.</td>
</tr>
<tr>
<td>Longitude</td>
<td>[Output Double] The X coordinate of the returned point.</td>
</tr>
<tr>
<td>Latitude</td>
<td>[Output Double] The Y coordinate of the returned point.</td>
</tr>
</tbody>
</table>
Returns
This call returns TRUE if successful, FALSE if not. The LastErrorCode property can be used to
determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error
Codes, for a full list of error codes.

GeocodeGetServerVersion()

Purpose
This method returns the version of the server.

Note: This method is not for use with VBScript. The appropriate call is

Syntax
GeocodeGetServerVersion(pVersionNum As Double) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVersionNum</td>
<td>[Output Double] MapMarker version in integer format (i.e., 9)</td>
</tr>
</tbody>
</table>

Returns
TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the
nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full
list of error codes.

GeocodeGetStatesFound()

Purpose
This method returns a list of states in the Address Dictionary path.

Note: This method is not for use with VBScript or ColdFusion. The appropriate calls are
GeocodeGetStatesFoundV() (page 215) and GeocodeGetStatesFound_A() (page 195),
respectively.

Syntax
GeocodeGetStatesFound(pstateList As String) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns
TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the
nature of any errors that occur. See MapMarker GeoEngine Error Codes on page 395 for a full
list of error codes.
**GeocodeGetStatesLicensed()**

**Purpose**
This method returns a list of licensed states.

**Note:** This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodeGetStatesLicensedV() (page 215) and GeocodeGetStatesLicensed_A() (page 196), respectively.

**Syntax**

```
GeocodeGetStatesLicensed(pstateList As String) As Boolean
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pstateList</td>
<td>[Output String] A list of space-separated 2-digit state abbreviations represent-</td>
</tr>
<tr>
<td></td>
<td>ing licensed states.</td>
</tr>
</tbody>
</table>

**Returns**

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

**GeocodeHwyExit()**

**Purpose**
This method uses input criteria of highway exit and state, and returns a count of the number of Hwy records that matched the input criteria. A subsequent call to GeocodeGetHwyExitCandidate() retrieves the matched candidates.

**Syntax**

```
GeocodeHwyExit(
    Hwyexit as String,
    State as String,
    count as Integer)As Boolean
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hwyext</td>
<td>[Input String] The highway and exit information in the form: <code>&lt;highway name&gt; &lt;highway directional&gt; EXIT&lt;exit number&gt;&lt;exit number suffix&gt;TO +&lt;exit name&gt;</code>. Where: highway name is the name of the highway name including type prefix or type suffix (e.g., Highway 8, I-90, or Pennsylvania Turnpike. highway directional is the direction of the highway, full or abbreviated (e.g., E or East). Optional. EXIT keyword is a case-insensitive keyword used to separate the highway information from the exit information. Required. exit number is the actual number of the exit. This is optional if there is an exit name in the input. exit number suffix is any non-numerical exit suffix (e.g., Exit 2E). Optional, not used by MapMarker. TO keyword is a case-insensitive keyword used to separate the exit name and exit number. Optional. exit name is the name of the exit (e.g., State Hwy 5). Optional, MapMarker will match only to exact names when there is no exit number in the input or there is no match to the exit number in the input. Sample input: I-87 EXIT 2E TO State Hwy 5 I-87 EXIT State Hwy 5 I-87 EXIT 2E I-87 EXIT 2 I-87 EXIT 2E State Hwy This input is used by the engine to look up the specified highway and exit in the highway exit data table (hwyexits.dbf in the Address Dictionary.</td>
</tr>
<tr>
<td>State</td>
<td>[Input String] Contains the state name, either in full or abbreviated.</td>
</tr>
<tr>
<td>count</td>
<td>[Output Integer] The total number of HwyExit candidates that matched.</td>
</tr>
</tbody>
</table>

Returns

This call returns TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GeocodeIsCandidateMultiUnit()

Purpose

This method determines if a candidate address specified by a zero-based index contains multiple units.

Syntax

```
GeocodeIsCandidateMultiUnit(geocodeHandle As long, index as Integer,
IsMultiUnit As Boolean) As Boolean
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] The handle returned from the Geocode call used.</td>
</tr>
<tr>
<td>index</td>
<td>[Input] Index value of the candidate to examine. This index is zero-based.</td>
</tr>
<tr>
<td>IsMultiUnit</td>
<td>[Output] True/False value indicating whether the candidate address has multiple units.</td>
</tr>
</tbody>
</table>

When using the GeocodeIsCandidateMultiUnit() function with the CAddressList, keep in mind that the index for GeocodeIsCandidateMultiUnit() is zero-based, and the index for CAddressList is one-based.

Returns

TRUE if successful; FALSE if not.

GeocodePostalCentroid()

Purpose

This call can be used to geocode a ZIP Code or ZIP+4 centroid.

Note: This method is not for use with VBScript or ColdFusion. The appropriate calls are GeocodePostalCentroidV() (page 217) and GeocodePostalCentroid_A() (page 198), respectively.

Syntax

GeocodePostalCentroid(Zip As String,
Plus4 As String,
Longitude As Double,
Latitude As Double,
Precision As Integer,
ResultCode As String) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zip</td>
<td>[Input String] ZIP Code address component</td>
</tr>
<tr>
<td>Plus4</td>
<td>[Input String] ZIP Code add-on address component</td>
</tr>
<tr>
<td>Longitude</td>
<td>[Output Double] Returns the longitude value.</td>
</tr>
<tr>
<td>Latitude</td>
<td>[Output Double] Returns the latitude value.</td>
</tr>
<tr>
<td>Precision</td>
<td>[Output Integer] Returns one of the following: 40 (Point ZIP); 30 (street-level match for an intersection address); 20 (street-level match for street address); 10 (Shape Path Centroid match); 3 (ZIP+4 Centroid match); 2 (ZIP+2 Centroid match); or 1 (ZIP Code Centroid match).</td>
</tr>
<tr>
<td>ResultCode</td>
<td>[Output String] Returns a value that is analogous to the GeoResult codes returned by MapMarker (defined in Chapter 3: Geocoding with MapMarker.)</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastError Code property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GeocodePostalCentroidWithSerial()

Purpose

This routine is called to geocode a ZIP Code centroid when the server requires that a serial number be provided to allow geocoding. This method is for OEM users only and it requires an additional program to create the OEM licenses (purchased separately from MapInfo Corporation).

Note: This method is not for use with ColdFusion or VBScript. For ColdFusion, the appropriate call is GeocodePostalCentroidWithSerial_A() (page 199). For VBScript, the appropriate call is GeocodePostalCentroidV() with the optional parameter pSerialNumber specified (page 217).

Syntax

GeocodePostalCentroidWithSerial(Zip As String,
   Plus4 As String,
   Longitude As Double,
   Latitude As Double,
   Precision As Integer,
   ResultCode As String,
   SerialNumber As String) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zip</td>
<td>[Input String] ZIP Code address component</td>
</tr>
<tr>
<td>Plus4</td>
<td>[Input String] ZIP Code add-on address component</td>
</tr>
<tr>
<td>Longitude</td>
<td>[Output Double] Returns the longitude value.</td>
</tr>
<tr>
<td>Latitude</td>
<td>[Output Double] Returns the latitude value.</td>
</tr>
<tr>
<td>Precision</td>
<td>[Output Integer] Returns one of the following: 10 (Shape Path Centroid match); 20 (street-level match for street address); 30 (street-level match for an intersection address); 40 (Point ZIP) 1 (ZIP Code Centroid match). 2 (ZIP+2 Centroid match); or 3 (ZIP+4 Centroid match);</td>
</tr>
<tr>
<td>ResultCode</td>
<td>[Output String] Returns a value that is analogous to the Georesult codes returned by MapMarker (defined in Understanding Result Codes on page 62.)</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] Serial number required by server, created by an OEM license program sold separately.</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastError Code property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GetCandidateAt()

Purpose

This call returns the text associated with a specific candidate in the Match Candidates list box.

Note: This method is not for use with VBScript. The appropriate call is GetCandidateAtV() (page 218).

Syntax

GetCandidateAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns

The string of tab-delimited text associated with the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GetCandidateCensusBlockIDAt()

Purpose

This call returns the Census Block ID of an address in the Match Candidates list box. The Census Block ID is a code of up to 15 digits and characters that describes the smallest of U.S. Census Bureau census units.

Note: This method is not for use with VBScript. The appropriate call is GetCandidateCensusBlockIDAtV() (page 219).

Syntax

GetCandidateCensusBlockIDAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns

The Census Block ID of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GetCandidateCityAt()

Purpose
This call returns the city portion of an address in the Match Candidates list box.

Note: This method is not to be used with VBScript. The appropriate call is GetCandidateCityAtV() (page 219).

Syntax
GetCandidateCityAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The city of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GetCandidateFirmAt()

Purpose
This call returns the firm portion of an address to the Match Candidates list box.

Note: This method is not for use with VBScript. The appropriate call is GetCandidateFirmAtV() (page 219).

Syntax
GetCandidateFirmAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The firm of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GetCandidateLatitudeAt()

Purpose
This call returns the latitude of an address in the Match Candidates list box.

Note: This method is not for use with VBScript. The appropriate call is GetCandidateLatitudeAtV() (page 220).

Syntax
GetCandidateLatitudeAt(Index As Integer) As Double

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The latitude of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GetCandidateLongitudeAt()

Purpose
This call returns the longitude of an address in the Match Candidates list box.

Note: This method is not for use with VBScript. The appropriate call is GetCandidateLongitudeAtV() (page 220).

Syntax
GetCandidateLongitudeAt(Index As Integer) As Double

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The longitude of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GetCandidatePlus4At()

Purpose
This call returns the ZIP Add-on portion of an address in the Match Candidates list box.

Note: This method is not for use with VBScript. The appropriate call is GetCandidatePlus4AtV() (page 221).

Syntax
GetCandidatePlus4At(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The ZIP Add on (Plus 4) of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GetCandidatePrecisionAt()

Purpose
This call returns the precision portion of an address in the Match Candidates list box. The precision defines the type of match: street level, shape path, intersection, Point ZIP, or ZIP centroid (either ZIP+4, ZIP+2 or ZIP Code).

Note: This method is not for use with VBScript. The appropriate call is GetCandidatePrecisionAtV() (page 221).

Syntax
GetCandidatePrecisionAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The precision of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GetCandidateResultCodeAt()

Purpose
Returns the result code for an address in the Match Candidates list box. It represents the type of match (single, multiple, or ZIP centroid) and how precisely the GeoEngine matched to the address components (house number, street name, prefix, street type, city name, ZIP Code, and whether it matched to the Address Dictionary or user-defined dictionary.)

Note: This method is not for use with VBScript. The appropriate call is GetCandidateResultCodeAtV() (page 221).

Syntax
GetCandidateResultCodeAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The result code of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GetCandidateStateAt()

Purpose
This call returns the state portion of an address in the Match Candidates list box.

Note: This method is not for use with VBScript. The appropriate call is GetCandidateStateAtV() (page 222).

Syntax
GetCandidateStateAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The state of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
GetCandidateStreetAt()

Purpose
This call returns the street portion of an address in the Match Candidates list box.

Note: This method is not for use with VBScript. The appropriate call is GetCandidateStreetAtV() (page 222).

Syntax
GetCandidateStreetAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The street address of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

GetCandidateZIPAt()

Purpose
This call returns the ZIP Code portion of an address in the Match Candidates list box.

Note: This method is not for use with VBScript. The appropriate call is GetCandidateZIPAtV() (page 223).

Syntax
GetCandidateZIPAt(Index As Integer) As String

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The ZIP Code of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.
**GetFullName()**

**Purpose**

This method returns the full name and path of the application.

**Syntax**

```
GetFullName() As String
```

**Returns**

The LastErrorCode property can be used to determine the nature of any errors that occur. See *Appendix H: MapMarker GeoEngine Error Codes*, for a full list of error codes.

---

**GetName()**

**Purpose**

This call returns the name of the application.

**Syntax**

```
GetName() As String
```

**Returns**

The LastErrorCode property can be used to determine the nature of any errors that occur. See *Appendix H: MapMarker GeoEngine Error Codes*, for a full list of error codes.

---

**GetVersionNum()**

**Purpose**

This function returns the version number of the OCX.

**Syntax**

```
GetVersionNum() As String
```

**Returns**

Version number of the OCX. The LastErrorCode property can be used to determine the nature of any errors that occur. See *Appendix H: MapMarker GeoEngine Error Codes*, for a full list of error codes.

---

**RefreshDialog()**

**Purpose**

This call redraws the MapMarker Geocoder Control's interface.

**Syntax**

```
RefreshDialog()
```

**Returns**

The LastErrorCode property can be used to determine the nature of any errors that occur. See *Appendix H: MapMarker GeoEngine Error Codes*, for a full list of error codes.
SelectCandidateAt()

Purpose
This function causes the MapMarker Geocoder Control to highlight the specified candidate in the candidate list box.

Syntax
SelectCandidateAt(Index As Integer) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The LastErrorCode property can be used to determine the nature of any errors that occur. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

OLE Automation Objects

There are two types of OLE Automation Objects available for you to use: CAddressList and CAddress.

CAddressList
The CAddress List collection is a pointer to a collection class that can be used to iterate through the candidate addresses. The CAddressList is a collection of CAddress objects that are returned by GeocodeGetCandidates().
CAddress

Each CAddress object has the following properties:

- Firm As String – The matched firm name
- Street As String – The matched street address
- City As String – The matched city
- State As String – The matched state
- Zip As String – The matched ZIP Code
- plus4 As String – The matched ZIP Add on
- Precision As Integer – The precision of the matched coordinate
  - 40 Point ZIP
  - 30 Street Level (Intersection)
  - 20 Street Level
  - 10 Shape Path Centroid
  - 3 ZIP+4 Centroid
  - 2 ZIP +2 Centroid
  - 1 ZIP Code centroid
- ResultCode As String – The matched result code
- Longitude As Double – The longitude of the matched address
- Latitude As Double – The latitude of the matched address
- CensusBlock As String – Census Block ID of the matched candidate.
- PrimaryStreet – The candidate primary street, if it has one.
- RecordType – The candidate ZIP+4 record type. The candidate may not have a record type if the match is from a TIGER record.
- DeliveryPoint – The candidate delivery point.
- CarrierRoute – The candidate carrier route.
- CheckDigit – The candidate check digit.
- TabbedAddress – The candidate tabbed address.
- Lacs – Single character output code that indicates that the record can be converted from a rural route address to a city-style address by using a USPS product, known as the Locatable Address Conversion Service.
- Pmb – “Private Mailbox” used at Commercial Mail Receiving Agencies (CMRA) such as Mailboxes Etc.
- PmbRange – The candidate range/number, when there is a PMB range in the input.
- AddressType – MapMarker addresstype.
- DpvConfCode – The DPV return code for the candidate. The return code can be one of the following:
  - N–This address does not exist.
  - Y–This address exists.
  - S–The street address exists, but the unit does not.
  - D–Address is incomplete (highrise with no unit, or rural route with no box number).
- DpvCmra – Indicates if the candidate belongs to a Commercial Mail Receiving Agency (CMRA).
- DpvFalsePos – If true, DPV has detected a condition in which the candidate appears to be artificially generated and not a legitimately obtained address. DPV will shut down immediately in this instance.
• DpvFn1 – Field that may be populated with a standard USPS footnote code. Each of the codes is described below.
• DpvFn2 – Field that may be populated with a standard USPS footnote code. Each of the codes is described below.
• DpvFn3 – Field that may be populated with a standard USPS footnote code. Each of the codes is described below.
• DpvFn4 – Field that may be populated with a standard USPS footnote code. Each of the codes is described below.
• DpvFn5 – Field that may be populated with a standard USPS footnote code. Each of the codes is described below.
• DpvFn6 – Field that may be populated with a standard USPS footnote code. Each of the codes is described below.

Footnote Codes

AA–Input address matched to the ZIP+4 file.
A1–Input address not matched to the ZIP+4 file.
BB–Input address matched to DPV (all components).
CC–Input address primary number matched to DPV but secondary number not matched (present but invalid).
N1–Input address primary number matched to DPV but highrise address missing secondary number.
M1–Input address primary number missing.
M3–Input address primary number invalid.
P1–Input address missing PO, RR, or HC box number.
RR–Input address matched to CMRA.
R1–Input address matched to CMRA but secondary number not present.
Programming Usage Example

The following is a sample user interface and corresponding Visual Basic code that illustrates the functions and usage of a geocoding control object. This VB project sample is found in \geogeng\samples\vb\ocxtest2.vbp after installation.
Dim objMM As Object 'MapMarker
Dim lngEngHandle&
Dim status As Integer
Dim numCandidates As Integer
Dim numCloseCandidates As Integer
Dim bConnected As Boolean
Dim bSettingModified As Boolean

Private Sub chkExtendedSearch_Click()
    bSettingModified = True
End Sub

Private Sub chkInState_Click()
    bSettingModified = True
End Sub

Private Sub chklastline_Click()
    If chklastline.Value = 1 Then
        txtlastline.Visible = True
        Label21.Visible = True
        txtCity.Visible = False
        txtState.Visible = False
        txtZip.Visible = False
        txtZip4.Visible = False
        Label5.Visible = False
        Label6.Visible = False
        Label7.Visible = False
        Label8.Visible = False
    Else
        txtlastline.Visible = False
        Label21.Visible = False
        txtCity.Visible = True
        txtState.Visible = True
        txtZip.Visible = True
        txtZip4.Visible = True
        Label5.Visible = True
        Label6.Visible = True
        Label7.Visible = True
        Label8.Visible = True
    End If
End Sub

Private Sub chkMatchCentroid_Click()
    bSettingModified = True
End Sub
Private Sub chkMatchHouse_Click()
    bSettingModified = True
End Sub

Private Sub chkMatchStreet_Click()
    bSettingModified = True
End Sub

Private Sub chkMatchXsec_Click()
    bSettingModified = True
End Sub

Private Sub cmdCentroid_Click()
    Dim retVal As Boolean
    Dim dblLong#, dblLat#, intPrec%, strResult$
    lblStatus = ""
    lblCand = ""
    lstCandidates.Clear
    On Error GoTo GeocodePostalCentroid_Error
    retVal = objMM.GeocodePostalCentroid(txtZip, txtZip4, dblLong, dblLat,
        intPrec, strResult)
    On Error GoTo 0
    If Not retVal Then
        errorCode = objMM.LastErrorCode
        If errorCode = 14 Then
            lblStatus = "Invalid address"
            'Sometimes .Connect returns true even when server is
            'not running, but .GeocodeAddress will return 1753
        ElseIf errorCode = 1753 Then
            If txtServer = "" Then
                lblStatus = "Local server is not running"
            Else
                lblStatus = "Server : " & txtServer & " is not
                running"
            End If
        Else
            lblStatus = "Error geocoding, error code: " &
            Str$(errorCode)
        End If
    Else
        lblLong = dblLong
        lblLat = dblLat
        Select Case intPrec
            Case 3
                lblPrec = "Zip+4 
            Case 2
                lblPrec = "Zip+2 
            Case 1
                lblPrec = "Zip centroid 
        End Select
        ' Centroid geocoding does not create candidates
        cmdGetCand.Enabled = False
        Exit Sub
    GeocodePostalCentroid_Error:
        MsgBox "Error Zip centroid geocoding"
End Sub
Private Sub cmdConnect_Click()
    Dim retVal As Boolean
    Dim serverVersion As Double
    Dim txtBuf As String
    If bConnected = False Then
        On Error GoTo Connect_Error
        retVal = objMM.Connect(txtServer)
        txtDbAvailable = objMM.GeocodeCheckDbAvailability
        If retVal = True Then
            bConnected = True
            ' Only when connected, can do geocoding
            cmdGeocode.Enabled = True
            cmdCentroid.Enabled = True
            cmd Disconnect.Enabled = True
            cmdConnect.Enabled = False
            DatabaseTypes = objMM.DatabaseTypes
        Else
            DatabaseTypes = 0
            GoTo Connect_Error
        End If
        retVal = objMM.GeocodeGetServerVersion(serverVersion)
        txtVersion = Str$(serverVersion)
        retVal = objMM.GeocodeGetStatesLicensed(txtBuf)
        txtLicensed = txtBuf
        retVal = objMM.GeocodeGetStatesFound(txtBuf)
        txtAvailable = txtBuf
    End If
    cmdRefresh_Click
    Exit Sub

Connect_Error:
    If txtServer = "" Then
        MsgBox ("Error connecting to local server")
    Else
        MsgBox ("Error connecting to server: " & txtServer)
    End If
End Sub

Private Sub cmdDisconnect_Click()
    Dim retVal As Boolean
    If bConnected = True Then
        retVal = objMM.GeocodeFreeSet(lngEngHandle)
        retVal = objMM.Disconnect
        bConnected = False
        cmdGeocode.Enabled = False
        cmdCentroid.Enabled = False
        cmdGetCand.Enabled = False
        cmdConnect.Enabled = True
        cmdDisconnect.Enabled = False
        cmdFreeSet.Enabled = False
        DatabaseTypes = objMM.DatabaseTypes
    End If
    cmdRefresh_Click
End Sub
Private Sub cmdFreeSet_Click()
    Dim retVal As Boolean
    retVal = objMM.GeocodeFreeSet(lngEngHandle)
    'Candidates freed
    cmdGetCand.Enabled = False
    cmdFreeSet.Enabled = False
End Sub

Private Sub cmdGeocode_Click()
    Dim retVal As Boolean
    Dim retCode
    Dim errorCode&
    ' User changed geocoding config parameters
    If bSettingModified = True Then
        retCode = MsgBox("Config parameters have been changed. Do you want to use new parameters?", 3, "Street Geocoding")
        If retCode = vbYes Then
            cmdSet_Click
        ElseIf retCode = vbCancel Then
            Exit Sub
        End If
    End If
    lblLong = ""
    lblLat = ""
    lblPrec = ""
    lblStatus = ""
    lblCand = ""
    lstCandidates.Clear
    On Error GoTo GeocodeAddress_Error
    If chklastline.Value = 1 Then
        retVal = objMM.GeocodeAddressLastline(lngEngHandle, txtFirm, txtStreet, txtlastline, status, numCandidates, numCloseCandidates)
    Else
        retVal = objMM.GeocodeAddress(lngEngHandle, txtFirm, txtStreet, txtCity, txtState, txtZip, status, numCandidates, numCloseCandidates)
    End If
    On Error GoTo 0
    If Not retVal Then
        errorCode = objMM.LastErrorCode
        If errorCode = 14 Then
            lblStatus = "Invalid address"
            'Sometimes .Connect returns true even when server is 'not running, but .GeocodeAddress will return 1753
        ElseIf errorCode = 1753 Then
            If txtServer = "" Then
                lblStatus = "Local server is not running"
            Else
                lblStatus = "Server : " & txtServer & " is not running"
            End If
        Else
            lblStatus = "Error geocoding, error code: " & Str$(errorCode)
        End If
    End If
Else
DatabaseTypes = objMM.DatabaseTypes
Select Case status
    Case 0
        lblStatus = "Single match"
    Case 1
        lblStatus = "Multiple match"
    Case 2
        lblStatus = "No close matches"
    Case 3
        lblStatus = "No candidates"
    Case 4
        lblStatus = "Single intersection match"
    Case 5
        lblStatus = "Multiple intersection match"
    Case 6
        lblStatus = "No close intersection matches"
    Case 7
        lblStatus = "No intersection candidates"
    Case 8
        lblStatus = "Possible intersections"
    Case Else
        lblStatus = Str$(status)
End Select
If numCandidates > 0 Then
    lblCand = Str$(numCandidates)
    'Now .GetCandidates can be called to get
    'candidate information
    cmdGetCand.Enabled = True
    cmdFreeSet.Enabled = True
Else
    cmdGetCand.Enabled = False
    cmdFreeSet.Enabled = False
End If
End If
Exit Sub
GeocodeAddress_Error:
    lblStatus = "Error geocoding"
End Sub
Private Sub cmdGetCand_Click()
Dim adrList As CAddressList
Dim adr As CAddress
Dim strAddr$, strCoords$, strPrec$, strCensId$
On Error GoTo GeocodeGetCandidates_Error
Set adrList = objMM.GeocodeGetCandidates(lngEngHandle,
    numCandidates)
On Error GoTo 0
'Fill in the ListBox with the candidates
lstCandidates.Clear
For Each adr In adrList
    With adr
        If IsEmpty(.Firm) Or .Firm = "" Then
        Else
        End If
        strCoords = " (" & Format(.Latitude, "##0.0000") & ", " & Format(.Longitude, "##0.0000") & ")"
    Select Case .Precision
        Case 30
            strPrec = " Street-level (Xsect) "
        Case 20
            strPrec = " Street-level "
        Case 10
            strPrec = " Shape-path Cent. "
        Case 3
            strPrec = " Zip+4 "
        Case 2
            strPrec = " Zip+2 "
        Case 1
            strPrec = " Zip centroid "
    End Select
    strCensId = .CensusBlock
    End With
    Set adr = Nothing
    lstCandidates.AddItem (strAddr & strCoords & strPrec & strCensId)
Next
Set adrList = Nothing
Exit Sub
GeocodeGetCandidates_Error:
    MsgBox "Error getting candidates"
End Sub
Private Sub cmdRefresh_Click()
    'Set UI from the properties of objMM
    With objMM
        txtServer = .ServerName
        lblBinding = .StringBinding
        ExeName = .GetName
        If .ExactHouse = True Then
            chkMatchHouse.Value = 1
        Else
            chkMatchHouse.Value = 0
        End If
        If .ExactName = True Then
            chkMatchStreet.Value = 1
        Else
            chkMatchStreet.Value = 0
        End If
        If .ExactZip = True Then
            chkMatchCentroid.Value = 1
        Else
            chkMatchCentroid.Value = 0
        End If
        If .ExpandSearch = True Then
            chkExtendedSearch.Value = 1
        Else
            chkExtendedSearch.Value = 0
        End If
        If .ExpandSearchInState = True Then
            chkInState.Value = 1
        Else
            chkInState.Value = 0
        End If
        If .MatchIntersections = True Then
            chkMatchXsec.Value = 1
        Else
            chkMatchXsec.Value = 0
        End If
        txtDistance = .ExpandDistance
        txtOffsetLine = .LinearOffset
        txtOffsetPerp = .PerpendicularSetback
        bSettingModified = False
    End With
End Sub
Private Sub cmdSet_Click()
    'Set the properties of objMM from values in the UI
    With objMM
        .ServerName = txtServer
        If chkMatchHouse.Value = 1 Then
            .ExactHouse = True
        Else
            .ExactHouse = False
        End If
        If chkMatchStreet.Value = 1 Then
            .ExactName = True
        Else
            .ExactName = False
        End If
        If chkMatchCentroid.Value = 1 Then
            .ExactZip = True
        Else
            .ExactZip = False
        End If
        If chkExtendedSearch.Value = 1 Then
            .ExpandSearch = True
        Else
            .ExpandSearch = False
        End If
        If chkInState.Value = 1 Then
            .ExpandSearchInState = True
        Else
            .ExpandSearchInState = False
        End If
        If chkMatchXsec.Value = 1 Then
            .MatchIntersections = True
        Else
            .MatchIntersections = False
        End If
        .ExpandDistance = txtDistance
        .LinearOffset = txtOffsetLine
        .PerpendicularSetback = txtOffsetPerp
        bSettingModified = False
    End With
End Sub
Private Sub Form_Load()
    Set objMM = CreateObject("MAPMARKR.MapMarkrCtrl.1")
    bConnected = False
    bSettingModified = False
    cmdGeocode.Enabled = False
    cmdCentroid.Enabled = False
    cmdGetCand.Enabled = False
    cmdFreeSet.Enabled = False
    cmdDisconnect.Enabled = False
    txtlastline.Visible = False
    Label21.Visible = False
    With objMM
        txtServer = .ServerName
        lblBinding = .StringBinding
        ExeName = .GetName
        If .ExactHouse = True Then
            chkMatchHouse.Value = 1
        Else
            chkMatchHouse.Value = 0
        End If
        If . ExactName = True Then
            chkMatchStreet.Value = 1
        Else
            chkMatchStreet.Value = 0
        End If
        If . ExactZip = True Then
            chkMatchCentroid.Value = 1
        Else
            chkMatchCentroid.Value = 0
        End If
        If .ExpandSearch = True Then
            chkExtendedSearch.Value = 1
        Else
            chkExtendedSearch.Value = 0
        End If
        If .ExpandSearchInState = True Then
            chkInState.Value = 1
        Else
            chkInState.Value = 0
        End If
        If . MatchIntersections = True Then
            chkMatchXsec.Value = 1
        Else
            chkMatchXsec.Value = 0
        End If
        txtDistance = .ExpandDistance
        txtOffsetLine = .LinearOffset
        txtOffsetPerp = .PerpendicularSetback
    End With
End Sub
Private Sub Option1_Click()
    objMM.NotUsingServer = False
    cmdConnect.Enabled = True
    cmdGeocode.Enabled = False
    cmdCentroid.Enabled = False
    txtServer.Enabled = True
End Sub

Private Sub Option2_Click()
    Dim serverVersion As Double
    Dim txtBuf As String
    objMM.NotUsingServer = True
    cmdConnect.Enabled = False
    cmdGeocode.Enabled = True
    cmdCentroid.Enabled = True
    txtServer.Enabled = False
    txtDbAvailable = objMM.GeocodeCheckDbAvailability
    DatabaseTypes = objMM.DatabaseTypes
    retVal = objMM.GeocodeGetServerVersion(serverVersion)
        txtVersion = Str$(serverVersion)
    retVal = objMM.GeocodeGetStatesLicensed(txtBuf)
    txtLicensed = txtBuf
    retVal = objMM.GeocodeGetStatesFound(txtBuf)
    txtAvailable = txtBuf
    ExeName = objMM.GetName
End Sub

Private Sub txtDistance_Change()
    bSettingModified = True
End Sub

Private Sub txtOffsetLine_Change()
    bSettingModified = True
End Sub

Private Sub txtOffsetPerp_Change()
    bSettingModified = True
End Sub
OLE Automation API for ColdFusion

This chapter explains the ColdFusion OLE Automation API. Use this to build your own geocoding control for your client application instead of using the program-ready MapMarker Geocoder Control (OCX) described in Chapter 10: OLE Automation API.

In this chapter:

- OLE Automation Methods for ColdFusion ................................. 186
- OLE Automation Objects ......................................................... 200
OLE Automation Methods for ColdFusion

Certain methods included for the OLE Automation API are not suitable when developing with ColdFusion. We have provided a set of methods for use while developing with ColdFusion. These methods do not use output parameters for returning data because ColdFusion does not have this capability.

These methods return an object of the class CGeoOut. Each function sets only the properties in CGeoOut that are relevant to its function call. Each function call listed is for use in the OLE Automation API and is not included in the MapMarker OCX. These methods are ColdFusion-specific versions of the methods described in Chapter 10: OLE Automation API.

- **GeocodeAddress_A()**
  Attempt to geocode an address and build a list of candidates. A subsequent call to GeocodeGetCandidates() returns all the needed candidate information from the GeocodeAddress_A() call.

- **GeocodeAddressEx_A()**
  Attempts to geocode an address and build a list of candidates. Also enables you to pass in street2 information as a separate field. A subsequent call to GeocodeGetCandidates() returns all the needed candidate information from the GeocodeAddressEx_A() call.

- **GeocodeAddressLastLine_A()**
  Allows you to enter an input address with an unparsed line containing city, state, and/or ZIP Code. A subsequent call to GeocodeGetCandidates() returns all the needed candidate information from the GeocodeAddressLastLine_A() call.

- **GeocodeAddressLastLineEx_A**
  Allows you to enter an input address with an unparsed last line containing city, state, and/or ZIP Code. Also enables you to pass in street2 information in a separate field. A subsequent call to GeocodeGetCandidates() returns all the needed candidate information from the GeocodeAddressLastLineEx_A() call.

- **GeocodeAddressWithSerial_A()**
  Requires user to input a serial number for geocoding. A subsequent call to GeocodeGetCandidates() returns all the needed candidate information from the GeocodeAddressWithSerial_A() call.

- **GeocodeAddressWithSerialEx_A()**
  Requires user to input a serial number for geocoding. Also enables you to pass in street2 information in a separate field. A subsequent call to GeocodeGetCandidates() returns all the needed candidate information from the GeocodeAddressWithSerialEx_A() call.

- **GeocodeFreeSet_A()**
  Frees the server side information after the GeocodeAddress() call.

- **GeocodeGetErrorText_A()**
  Returns a short text description of the specified error.

- **GeocodeGetHwyExitCandidate_A()**
  Geocodes a highway exit following call to GeocodeHwyExit_A().

- **GeocodeGetStatesFound_A()**
  Returns a list of 2-digit state abbreviations representing states found in the Address Dictionary path. These are not necessarily licensed states.
GeocodeGetStatesLicensed_A() .......................... 196
  Returns a list of 2-digit state abbreviations representing licensed states.

GeocodeHwyExit_A() ........................................ 196
  This method returns a count of the number of highway exit records based on input
criteria of highway exit and state.

GeocodeIsCandidateMultiUnit_A() ......................... 198
  Returns a Yes/No value indicating whether the candidate address contains multiple
units.

GeocodePostalCentroid_A() .......................... 198
  Geocodes a ZIP Code or ZIP+4 centroid.

GeocodePostalCentroidWithSerial_A() ..................... 199
  Requires a serial number for ZIP Code-level geocoding.

GeocodeAddress_A()

Purpose
  This is the main call for the OLE Automation API (for use with ColdFusion). This method attempts
to geocode an address and build a list of candidates. To get the list of candidates, you must call
GeocodeGetCandidates(). It returns the candidate information from the GeocodeAddress_A() call.

Syntax
  GeocodeAddress_A(firm as String,
                   street as String,
                   city as String,
                   state as String,
                   zip as String) as Object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>[Input String] Firm name to matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to matched.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
</tbody>
</table>

Returns
  This function returns an Object of class CGeoOut with the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoHandle</td>
<td>The geocoding handle.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>Number of close match candidates found.</td>
</tr>
</tbody>
</table>
GeocodeAddressEx_A()

**Purpose**
This method attempts to geocode an address and build a list of candidates. It works similarly to GeocodeAddress_A(); however, it enables you to pass in street2 information in a separate field.

To get the list of candidates, you must call GeocodeGetCandidates(). It returns all the needed candidate information from the GeocodeAddressEx_A() call.

**Syntax**
```java
Object GeocodeAddressEx_A(Firm as String,
                          Street as String,
                          Street2 as String,
                          City as String,
                          State as String,
                          Zip as String)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>street2</td>
<td>[Input String] Secondary address to be matched.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State name to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] ZIP Code (including ZIP + 4) to be matched.</td>
</tr>
</tbody>
</table>

**Returns**

This function returns an Object of class CGeoOut with the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoHandle</td>
<td>The geocoding handle.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>Number of close match candidates found.</td>
</tr>
<tr>
<td>matchStatus</td>
<td>Integer indicating type of match found: SINGLE_MATCH, MULTIPLE_MATCH, etc.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>
GeocodeAddressLastLine_A()

Purpose
This method allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database). To get the list of candidates, you must call GeocodeGetCandidates(). It returns all the needed candidate information from the GeocodeAddressLastLine_A() method.

Syntax
GeocodeAddressLastLine_A(firm As String, 
  street As String, 
  lastline As String, 
  SerialNumber As String) As Object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>lastline</td>
<td>[Input String] Unparsed last line or address to be matched.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Optional Input String ]Serial number required by server, created by an OEM license program sold separately.</td>
</tr>
</tbody>
</table>

Returns
This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoHandle</td>
<td>The geocoding handle.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>Number of close match candidates found.</td>
</tr>
<tr>
<td>matchStatus</td>
<td>Integer indicating type of match found: SINGLE_MATCH, MULTIPLE_MATCH, etc.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>
GeocodeAddressLastLineEx_A

Purpose
This method allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database). It works similarly to GeocodeAddressLastLine_A(); however, it also enables you to pass in street2 information in a separate field.

To get the list of candidates, you must call GeocodeGetCandidates(). It returns all the needed candidate information from the GeocodeAddressLastLineEx_A() call.

Syntax

Object GeocodeAddressLastLineEx_A(Firm as String, Street as String, Street2 as String, lastline as String, SerialNumber as VARIANT)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>street2</td>
<td>[Input String] Secondary address to be matched.</td>
</tr>
<tr>
<td>lastline</td>
<td>[Input String] Unparsed last line or address to be matched.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Optional Input String] Serial number required by server, created by an OEM license program sold separately.</td>
</tr>
</tbody>
</table>

Returns
This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoHandle</td>
<td>The geocoding handle.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>Number of close match candidates found.</td>
</tr>
<tr>
<td>matchStatus</td>
<td>Integer indicating type of match found: SINGLE_MATCH, MULTIPLE_MATCH, etc.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>
GeocodeAddressWithSerial_A()

Purpose
This routine is called to geocode an address when the server requires that a serial number be provided to allow geocoding. This method is for OEM users only and it requires an additional program to create the OEM licenses (purchased separately from MapInfo Corporation).

To get the list of candidates, you must call GeocodeGetCandidates(). It returns all the needed candidate information from the GeocodeAddressWithSerial_A() call.

Syntax

GeocodeAddressWithSerial_A(firm as String,  
street as String,  
city as String,  
state as String,  
zip as String,  
SerialNumber as String) as Object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] Serial number required by server, created by an OEM license program sold separately.</td>
</tr>
</tbody>
</table>

Returns

This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoHandle</td>
<td>The geocoding handle.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>Number of close match candidates found.</td>
</tr>
<tr>
<td>matchStatus</td>
<td>Integer indicating type of match found: SINGLE_MATCH, MULTIPLE_MATCH, etc.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>
GeocodeAddressWithSerialEx_A()

**Purpose**
This routine is called to geocode an address when the server requires that a serial number be provided to allow geocoding. This method is for OEM users only and it requires an additional program to create the OEM licenses (purchased separately from MapInfo Corporation). It works similarly to GeocodeAddressWithSerial_A; however, it enables you to pass in street2 information in a separate field.

To get the list of candidates, you must call GeocodeGetCandidates(). It returns all the needed candidate information from the GeocodeAddressWithSerialEx_A() call.

**Syntax**
```java
Object GeocodeAddressWithSerialEx_A(Firm as String,
                                         Street as String,
                                         Street2 as String,
                                         City as String,
                                         State as String,
                                         Zip as String,
                                         SerialNumber as String)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>street2</td>
<td>[Input String] Secondary address to be matched.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] Serial number required by server, created by an OEM license program sold separately.</td>
</tr>
</tbody>
</table>

**Returns**
This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoHandle</td>
<td>The geocoding handle.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>Number of close match candidates found.</td>
</tr>
</tbody>
</table>
GeocodeFreeSet_A()

Purpose
This call frees all the server side information maintained after the GeocodeAddress_A() call. The client should call this once they have finished with the address. Once the information is freed, the handle cannot be used again unless the same value is returned by a later call to GeocodeAddress_A().

GeocodeFreeSet_A() releases the memory structure allocated to hold the candidate list. Whenever GeocodeAddress_A() is invoked, a memory structure is allocated, the structure is populated with a list of possible candidates, and a context handle to that structure is returned to the caller.

MapMarker can allocate a maximum of 1,024 of these structures. To avoid running out of memory or blocking the MapMarker Server, call GeocodeFreeSet_A() as soon as you have finished evaluating the candidate list.

Syntax
GeocodeFreeSet_A(geohandle As Long) As Object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input Long] Used to reference information about the last geocoding operation.</td>
</tr>
</tbody>
</table>

Returns
This function returns an Object of class CGeoOut with the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoHandle</td>
<td>If the call is successful, the geocoding handle is reset to 0.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>
GeocodeGetErrorText_A()

Purpose
Given a valid MapMarker error code, this method returns a string giving a short text description of the error.

Syntax
GeocodeGetErrorText_A(errcode As Integer) As Object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errcode</td>
<td>[Input Integer] MapMarker error code.</td>
</tr>
</tbody>
</table>

Returns
This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorString</td>
<td>Text string associated with a given MapMarker error code.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>

GeocodeGetHwyExitCandidate_A()

Purpose
Given a zero-based index number, this method returns highway name, exit number, and/or exit name, as well as the ZIP Code, city, and coordinates of the exit location. This method cannot be used until its corresponding call, GeocodeHwyExit_A(), has been invoked. Using the count from that function, the individual candidates can be retrieved from GeocodeGetHwyExitCandidate_A(). The first candidate is 0.

Syntax
CGeoOut GeocodeGetHwyExitCandidate_A (index as integer)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>[Input Integer] Zero-based index of candidates to retrieve.</td>
</tr>
</tbody>
</table>
Returns

This method returns a CGeoOut object that can have the following fields populated. These fields may not have return values.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HwyTypePre</td>
<td>Pre-type of the matched highway.</td>
</tr>
<tr>
<td>HwyName</td>
<td>Name of the matched highway.</td>
</tr>
<tr>
<td>HwyTypeSuffix</td>
<td>Post-type of the matched highway.</td>
</tr>
<tr>
<td>HwyDir</td>
<td>Matched direction.</td>
</tr>
<tr>
<td>ExitNumber</td>
<td>Matched exit number.</td>
</tr>
<tr>
<td>ExitName</td>
<td>Matched exit name.</td>
</tr>
<tr>
<td>City</td>
<td>Returned city name.</td>
</tr>
<tr>
<td>State</td>
<td>Returned state name.</td>
</tr>
<tr>
<td>Zipcode</td>
<td>Returned ZIP Code.</td>
</tr>
<tr>
<td>Longitude</td>
<td>X coordinate of the returned point.</td>
</tr>
<tr>
<td>Latitude</td>
<td>Y coordinate of the returned point.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>

GeocodeGetStatesFound_A()

Purpose

This method returns a list of states in the Address Dictionary path.

Syntax

GeocodeGetStatesFound_A() As Object

Returns

This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StateList</td>
<td>A list of space-separated 2-digit state abbreviations.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>
**GeocodeGetStatesLicensed_A()**

**Purpose**
This method returns a list of licensed states.

**Syntax**

```csharp
GeocodeGetStatesLicensed_A() As Object
```

**Returns**
This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StateList</td>
<td>A list of space-separated 2-digit state abbreviations representing licensed states.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>

**GeocodeHwyExit_A()**

**Purpose**
This method uses input criteria of hwyexit and state, and returns a count of the number of Hwy records that matched the input criteria. A subsequent call to GeocodeGetHwyExitCandidate_A() retrieves the matched candidates.

**Syntax**

```csharp
CGeoOut GeocodeHwyExit_A(Hwyexit as String, State as String)
```
Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hwyext</td>
<td>[Input String] The highway and exit information in the form: <code>&lt;highway name&gt; &lt;highway directional&gt; EXIT&lt;exit number&gt;&lt;exit number suffix&gt;TO +=&lt;exit name&gt;</code>. Where: highway name is the name of the highway name including type prefix or type suffix (e.g., Highway 8, I-90, or Pennsylvania Turnpike). highway directional is the direction of the highway, full or abbreviated (e.g., E or East). Optional. EXIT keyword is a case-insensitive keyword used to separate the highway information from the exit information. Required. exit number is the actual number of the exit. This is optional if there is an exit name in the input. exit number suffix is any non-numerical exit suffix (e.g., Exit 2E). Optional, not used by MapMarker. TO keyword is a case-insensitive keyword used to separate the exit name and exit number. Optional. exit name is the name of the exit (e.g., State Hwy 5). Optional, MapMarker will match only to exact names when there is no exit number in the input or there is no match to the exit number in the input. Sample input: I-87 EXIT 2E TO State Hwy 5 I-87 EXIT State Hwy 5 I-87 EXIT 2E I-87 EXIT 2 I-87 EXIT 2E State Hwy . This input is used by the engine to look up the specified highway and exit in the highway exit data table (hwyexits.dbf in the Address Dictionary)</td>
</tr>
<tr>
<td>State</td>
<td>[Input String] Contains the state name, either in full or abbreviated.</td>
</tr>
</tbody>
</table>

Returns

This method returns a CGeoOut object that can have the following fields populated. These fields will always have return values.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
<tr>
<td>HwyCount</td>
<td>The total number of HwyExit candidates that matched.</td>
</tr>
</tbody>
</table>
GeocodeIsCandidateMultiUnit_A()

Purpose
This method determines whether a candidate address specified by a zero-based index contains multiple units.

Syntax
GeocodeIsCandidateMultiUnit_A(geocodeHandle As long, index As Integer) As Object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] The handle returned from the Geocode call used.</td>
</tr>
<tr>
<td>index</td>
<td>[Input] The index value of the candidate address to examine. The index is zero-based.</td>
</tr>
</tbody>
</table>

When using the GeocodeIsCandidateMultiUnit_A() function with the CAddressList, keep in mind that the index for GeocodeIsCandidateMultiUnit_A() is zero-based, and the index for CAddressList is one-based.

Returns
This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
<tr>
<td>MultiUnit</td>
<td>Yes/No value indicating whether the candidate address contains multiple units.</td>
</tr>
</tbody>
</table>

GeocodePostalCentroid_A()

Purpose
This call can be used to geocode a ZIP Code or ZIP+4 centroid.

Syntax
GeocodePostalCentroid A(Zip as String, ZipPlus4 as String) as Object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zip</td>
<td>[Input String] ZIP Code address component.</td>
</tr>
</tbody>
</table>
Returns

This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longitude</td>
<td>The longitude of geocoded point.</td>
</tr>
<tr>
<td>Latitude</td>
<td>The latitude of geocoded point.</td>
</tr>
<tr>
<td>Precision</td>
<td>Precision of the matched coordinate</td>
</tr>
<tr>
<td>ResultCode</td>
<td>The matched result code.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>

GeocodePostalCentroidWithSerial_A()

Purpose

This routine is called to geocode a ZIP Code centroid when the server requires that a serial number be provided to allow geocoding. This method is for OEM users only and it requires an additional program to create the OEM licenses (purchased separately from MapInfo Corporation).

Syntax

GeocodePostalCentroidWithSerial_A(Zip as String, ZipPlus4 as String, pSerialNumber as String) as Object

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zip</td>
<td>[Input String] ZIP Code address component.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Input String] Serial number required by server, created by an OEM license program sold separately.</td>
</tr>
</tbody>
</table>

Returns

This function returns an Object of class CGeoOut. It has the following properties:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longitude</td>
<td>The longitude of geocoded point.</td>
</tr>
<tr>
<td>Latitude</td>
<td>The latitude of geocoded point.</td>
</tr>
<tr>
<td>Precision</td>
<td>Precision of the matched coordinate</td>
</tr>
<tr>
<td>ResultCode</td>
<td>The matched result code.</td>
</tr>
<tr>
<td>retStatus</td>
<td>Success or failure of the function call.</td>
</tr>
</tbody>
</table>
OLE Automation Objects

The following OLE automation object is available: CGeoOut

CGeoOut

The CGeoOut object is a collection class that contains information returned by the ColdFusion methods. Each function sets only those properties that are relevant.

- geoHandle As Long – The geocoding handle.
- Latitude As Double – The latitude of geocoded point.
- Longitude As Double – The longitude of geocoded point.
- Precision As Integer – Precision of the matched coordinate.
- ResultCode As String – The matched result code.
- StateList As String – List of states (if returned from GeocodeGetStatesLicensed_A() then licensed states. If returned from GeocodeGetStatesFound() then found states).
- matchStatus As Integer – Indicates type of match found: SINGLE_MATCH, MULTIPLE_MATCH, etc.
- numCandidates As Integer – Total number of candidates found.
- numCloseCandidates As Integer – Number of close match candidates found.
- retStatus As Boolean – Success or failure of the function call.
- ErrorString As String – Text string associated with a given MapMarker error code.
- HwyTypePre As String– Type prefix of the matched highway.
- HwyName As String– The name of the matched highway.
- HwyTypeSuffix As String– Type suffix of the matched highway.
- HwyDir As String– The direction of the matched highway.
- ExitNumber As String– The number of the matched exit.
- ExitName As String– The name of the matched exit.
- City As String– The returned city name.
- State As String– The returned state name.
- Zipcode As String– The returned ZIP Code.
This chapter explains the ASP/VBScript OLE Automation API. Use this to build your own geocoding control for your client application instead of using the program-ready MapMarker Geocoder Control (OCX), described in Chapter 10: OLE Automation API.

In this chapter:

- ASP/VBScript Methods ................................................................. 202
- ASP Sample Application .............................................................. 224
ASP/VBScript Methods

Certain methods included for the OLE Automation API are not suitable when developing with ASP/VBScript. We have provided a set of methods for use while developing with ASP/VBScript. These methods use output parameters that have the capability to return variables of the type VARIANT.

The following methods are for use with ASP/VBScript. These methods are ASP/VBScript-specific versions of the methods described in Chapter 10: OLE Automation API.

- **DpvGeocodeAddressV()**  ................................................................. 204
  Turns DPV mode on, and then attempts to geocode an address and build a list of candidates. When used with the optional `pSerialNumber`, this method acts like `GeocodeAddressWithSerial()`.

- **DpvGeocodeAddressLastLineV()**  .................................................. 205
  Turns DPV mode on, and then allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database).

- **GeocodeAddressV()**  ................................................................. 207
  Attempts to geocode an address and build a list of candidates. When used with the optional `pSerialNumber`, this method acts like `GeocodeAddressWithSerial()`.

- **GeocodeAddressExV()**  .............................................................. 208
  Attempts to geocode an address and build a list of candidates. When used with the optional `pSerialNumber`, this method acts like `GeocodeAddressWithSerial()`. Also enables you to pass in street2 information as a separate field.

- **GeocodeAddressLastLineV()**  .................................................... 210
  Allows you to enter an input address with an unparsed line containing city, state, and/or ZIP Code.

- **GeocodeAddressLastLineExV()**  ................................................ 211
  Allows you to enter an input address with an unparsed line containing city, state, and/or ZIP Code, and enables you to pass in street2 information as a separate field.

- **GeocodeFreeSetV()**  ................................................................. 212
  Frees the server side information after the `GeocodeAddressV()` call.

- **GeocodeGetCandidatesV()**  ....................................................... 212
  Gets all the candidate information from the `GeocodeAddress()` call.

- **GeocodeGetErrorTextV()**  ....................................................... 213
  Returns a short text description of the specified error.

- **GeocodeGetHwyExitCandidateV()**  ......................................... 213
  Returns the highway exit candidate information after a call to `GeocodeHwyExitV()`.

- **GeocodeGetServerVersionV()**  .................................................. 215
  Returns the server version number.

- **GeocodeGetStatesFoundV()**  .................................................... 215
  Returns a list of 2-digit state abbreviations representing states found in the Address Dictionary path. These are not necessarily licensed states.

- **GeocodeGetStatesLicensedV()**  .................................................. 215
  Returns a list of 2-digit state abbreviations representing licensed states.

- **GeocodeHwyExitV()**  ............................................................... 216
  This method uses input criteria of highway exit and state, and returns a count of the number of highway records based on input criteria of highway exit and state.
- **GeocodeIsCandidateMultiUnitV()** ................................................................. 217
  Returns a True/False value indicating whether the candidate address contains multiple units.

- **GeocodePostalCentroidV()** ................................................................. 217
  Geocodes a ZIP Code or ZIP+4 centroid. This method when used with the optional pSerialNumber parameter acts like GeocodePostalCentroidWithSerial().

- **GeocodePostalCentroidWithSerial()** ....................................................... 217
  Geocodes a ZIP Code or ZIP+4 centroid with a serial number.

- **GetCandidateAtV()** .................................................................................. 218
  Returns the line of text in the Match Candidates List Box in OCX as a VARIANT.

- **GetCandidateCensusBlockIDAtV()** ............................................................ 219
  Returns the Census Block ID as a VARIANT.

- **GetCandidateCityAtV()** ............................................................................ 219
  Returns the city portion of the address as a VARIANT.

- **GetCandidateFirmAtV()** ............................................................................ 219
  Returns the firm portion of the address as a VARIANT.

- **GetCandidateLatitudeAtV()** ...................................................................... 220
  Returns the latitude as a VARIANT.

- **GetCandidateLongitudeAtV()** .................................................................... 220
  Returns the longitude as a VARIANT.

- **GetCandidatePlus4AtV()** .......................................................................... 221
  Returns the ZIP add-on of the address as a VARIANT.

- **GetCandidatePrecisionAtV()** .................................................................... 221
  Returns the precision for the match as a VARIANT. Precision refers to the quality of the match: to street level, shape path, intersection, Point ZIP, or ZIP centroid.

- **GetCandidateResultCodeAtV()** ................................................................. 221
  Returns the result code portion of the address as a VARIANT.

- **GetCandidateStateAtV()** ........................................................................... 222
  Returns the state portion of the address as a VARIANT.

- **GetCandidateStreetAtV()** .......................................................................... 222
  Returns the street portion of the address as a VARIANT.

- **GetCandidateZIPAtV()** ............................................................................. 223
  Returns the ZIP portion of the address as a VARIANT.

- **GetCandidatePostalCodeAtV()** .................................................................. 224
  Returns the ZIP Code portion of the address as a VARIANT.
DpvGeocodeAddressV()

Purpose
This method turns DPV mode on, and then attempts to geocode an address and build a list of candidates. This call also enables you to pass in street2 information as a separate field.

Note: The CASSMode property must be set to True or this method will be unable to enable DPV mode.

Syntax
DpvGeocodeAddressV(geocodeHandle As VARIANT, fir As String, street As String, street2 As String, city As String, state As String, zip As String, status As VARIANT, numCandidates As VARIANT, numCloseCandidates As VARIANT, pSerialNumber as String) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output VARIANT] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine what error occurred. See Appendix H: MapMarker GeoEngine Error Codes, for a complete list of errors.

DpvGeocodeAddressLastLineV()

Purpose

This method turns DPV mode on, and then allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database). This call also enables you to pass in street2 information as a separate field.

Note: The CASSMode property must be set to True or this method will be unable to enable DPV mode.
Syntax

DpvGeocodeAddressLastLineV(geocodeHandle As VARIANT,
    firm As String,
    street As String,
    street2 As String,
    lastline As String,
    status As VARIANT,
    numCandidates As VARIANT,
    numCloseCandidates As VARIANT,
    pSerialNumber As String) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output VARIANT] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>lastline</td>
<td>[Input String] Unparsed last line or address to be matched.</td>
</tr>
<tr>
<td>status</td>
<td>[Output VARIANT] One of the following:</td>
</tr>
<tr>
<td></td>
<td>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code (CASS mode only).</td>
</tr>
<tr>
<td></td>
<td>0 – SINGLE_MATCH – a single close match was found for the input address.</td>
</tr>
<tr>
<td></td>
<td>1 – MULTIPLE_MATCH – multiple match candidates were found and MapMarker could not choose between at least two of them.</td>
</tr>
<tr>
<td></td>
<td>2 – NO_MATCHES – candidates found, but none considered a close match.</td>
</tr>
<tr>
<td></td>
<td>3 – NO_CANDIDATES – no candidates found for the input address.</td>
</tr>
<tr>
<td></td>
<td>4 – SINGLE_INTERSECT_MATCH – an intersection match was found.</td>
</tr>
<tr>
<td></td>
<td>5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was found.</td>
</tr>
<tr>
<td></td>
<td>6 – NO_INTERSECT_MATCHES – no close intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>7 – NO_INTERSECT_CANDIDATES – no intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>8 – POSSIBLE_INTERSECTION – possible close intersection match found.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output VARIANT] Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output VARIANT] Number of close candidates.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Optional Input String] Serial number used for restricted geocoding.</td>
</tr>
</tbody>
</table>
Returns
TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine what error occurred. See Appendix H: MapMarker GeoEngine Error Codes, for a complete list of errors.

GeocodeAddressV()

Purpose
This is the main call for the OLE Automation API special ASP/VBScript methods. It attempts to geocode an address and build a list of candidates.

Syntax

GeocodeAddressV(geocodeHandle As VARIANT, firm As String, street As String, city As String, state As String, zip As String, status As VARIANT, numCandidates As VARIANT, numCloseCandidates As VARIANT, pSerialNumber as String) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output VARIANT] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine what error occurred. See Appendix H: MapMarker GeoEngine Error Codes, for a complete list of errors.

GeocodeAddressExV()

Purpose

This call attempts to geocode an address and build a list of candidates. It works similarly to GeocodeAddressV(); however, it also enables you to pass in street2 information as a separate field.

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>[Output VARIANT] One of the following:</td>
</tr>
<tr>
<td></td>
<td>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code</td>
</tr>
<tr>
<td></td>
<td>(CASS mode only).</td>
</tr>
<tr>
<td></td>
<td>0 – SINGLE_MATCH – a single close match was found for the input address.</td>
</tr>
<tr>
<td></td>
<td>1 – MULTIPLE_MATCH – multiple match candidates were found and MapMarker</td>
</tr>
<tr>
<td></td>
<td>could not choose between at least two of them.</td>
</tr>
<tr>
<td></td>
<td>2 – NO_MATCHES – candidates found, but none considered a close match.</td>
</tr>
<tr>
<td></td>
<td>3 – NO_CANDIDATES – no candidates found for the input address.</td>
</tr>
<tr>
<td></td>
<td>4 – SINGLE_INTERSECT_MATCH – an intersection match was found.</td>
</tr>
<tr>
<td></td>
<td>5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was</td>
</tr>
<tr>
<td></td>
<td>found.</td>
</tr>
<tr>
<td></td>
<td>6 – NO_INTERSECT_MATCHES – no close intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>7 – NO_INTERSECT_CANDIDATES – no intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>8 – POSSIBLE_INTERSECTION – possible close intersection match found.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output VARIANT] Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output VARIANT] Number of close candidates.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Optional Input String] Serial number used for restricted geocoding.</td>
</tr>
<tr>
<td></td>
<td>Causes the method to act like GeocodeAddressWithSerial().</td>
</tr>
</tbody>
</table>

Causes the method to act like GeocodeAddressWithSerial().
Syntax

```vba
GeocodeAddressExV(geocodeHandle as VARIANT,
    Firm as String,
    Street as String,
    Street2 as String,
    City as String,
    State as String,
    Zip as String,
    Status as VARIANT,
    NumCandidates as VARIANT,
    NumCloseCandidates as VARIANT,
    SerialNumber as VARIANT)As Boolean
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output VARIANT] Context handle needed to reference internal items. A handle is retrieved by the call.</td>
</tr>
<tr>
<td>Firm</td>
<td>[Input String] Firm name.</td>
</tr>
<tr>
<td>Street</td>
<td>[Input String] Street name.</td>
</tr>
<tr>
<td>Street2</td>
<td>[Input String] Secondary address.</td>
</tr>
<tr>
<td>City</td>
<td>[Input String] City name.</td>
</tr>
<tr>
<td>State</td>
<td>[Input String] State name.</td>
</tr>
<tr>
<td>Zip</td>
<td>[Input String] ZIP Code (includes ZIP + 4)</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output VARIANT] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output VARIANT] The number of close match candidates.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Optional Input VARIANT] The serial number for OEM license usage.</td>
</tr>
</tbody>
</table>

Returns

This call returns TRUE if successful, FALSE if not.
GeocodeAddressLastLineV()

Purpose
This method allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database).

Syntax
```
GeocodeAddressLastLineV(geocodeHandle As VARIANT,
firm As String,
street As String,
lastline As String,
status As VARIANT,
numCandidates As VARIANT,
numCloseCandidates As VARIANT,
pSerialNumber As String) As Boolean
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output VARIANT] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>lastline</td>
<td>[Input String] Unparsed last line or address to be matched.</td>
</tr>
<tr>
<td>status</td>
<td>[Output VARIANT] One of the following:</td>
</tr>
<tr>
<td></td>
<td>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code (CASS mode only).</td>
</tr>
<tr>
<td></td>
<td>0 – SINGLE_MATCH – a single close match was found for the input address.</td>
</tr>
<tr>
<td></td>
<td>1 – MULTIPLE_MATCH – multiple match candidates were found and MapMarker could not choose between at least two of them.</td>
</tr>
<tr>
<td></td>
<td>2 – NO_MATCHES – candidates found, but none considered a close match.</td>
</tr>
<tr>
<td></td>
<td>3 – NO_CANDIDATES – no candidates found for the input address.</td>
</tr>
<tr>
<td></td>
<td>4 – SINGLE_INTERSECT_MATCH – an intersection match was found.</td>
</tr>
<tr>
<td></td>
<td>5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was found.</td>
</tr>
<tr>
<td></td>
<td>6 – NO_INTERSECT_MATCHES – no close intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>7 – NO_INTERSECT_CANDIDATES – no intersection candidates found.</td>
</tr>
<tr>
<td></td>
<td>8 – POSSIBLE_INTERSECTION – possible close intersection match found.</td>
</tr>
</tbody>
</table>
Returns
TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine what error occurred. See Appendix H: MapMarker GeoEngine Error Codes, for a complete list of errors.

**GeocodeAddressLastLineExV()**

**Purpose**
This method allows you to enter an address with an unparsed last line (e.g., the city, state, and/or ZIP Code are contained in the same field in the database). This call works similarly to GeocodeAddressLastLineV(); however, it also enables you to pass in street2 information as a separate field.

**Syntax**

```vbnet
GeocodeAddressLastLineExV(geocodeHandle as VARIANT,
Firm as String,
Street as String,
Street2 as String,
lastline as String,
status as VARIANT,
NumCandidates as VARIANT,
NumCloseCandidates as VARIANT,
SerialNumber as VARIANT)As Boolean
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numCandidates</td>
<td>[Output VARIANT] Total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output VARIANT] Number of close candidates.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Optional Input String] Serial number used for restricted geocoding.</td>
</tr>
</tbody>
</table>

**Parameter Description**

- **geocodeHandle** [Output VARIANT] Context handle needed to reference internal items. A handle is retrieved by the call.
- **Firm** [Input String] Firm name.
- **Street** [Input String] Street name.
- **Street2** [Input String] Secondary address.
- **Lastline** [Input String] Lastline information (city, state, ZIP).
- **status** [Output VARIANT] The match status of the geocode.
- **NumCandidates** [Output VARIANT] The total number of candidates.
- **NumCloseCandidates** [Output VARIANT] The number of close match candidates.
- **SerialNumber** [Optional Input VARIANT] The serial number for OEM license usage.
Returns

This call returns TRUE if successful, FALSE if not.

---

**GeocodeFreeSetV()**

**Purpose**

This call frees all the server side information maintained after the `GeocodeAddressV()` call. The client should call this once they have finished with the address. Once the information is freed, the handle cannot be used again unless the same value is returned by a later call to `GeocodeAddressV()`.

`GeocodeFreeSetV()` releases the memory structure allocated to hold the candidate list. Whenever `GeocodeAddressV()` is invoked, a memory structure is allocated, the structure is populated with a list of possible candidates, and a context handle to that structure is returned to the caller.

MapMarker can allocate a maximum of 1,024 of these structures. To avoid running out of memory or blocking the MapMarker Server, call `GeocodeFreeSetV()` as soon as you have finished evaluating the candidate list.

**Syntax**

```
GeocodeFreeSetV(geocodeHandle As VARIANT) As Boolean
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input/Output VARIANT] Used to reference information about the geocoding operation in other calls. geocodeHandle is set to 0 if the function call is successful.</td>
</tr>
</tbody>
</table>

**Returns**

TRUE if successful, FALSE if not.

---

**GeocodeGetCandidatesV()**

**Purpose**

This call returns all the needed candidate information from the `GeocodeAddressV()` call. This information is stored in a collection class.

**Syntax**

```
GeocodeGetCandidatesV(geocodeHandle As VARIANT,
numCandidates As VARIANT) As Object
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input VARIANT] Used to reference information about the most recent geocoding operation.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Input VARIANT] Specifies the number of candidates that should be returned (i.e., the size of the collection).</td>
</tr>
</tbody>
</table>

Returns

A CAddressList collection of CAddress objects. The number of addresses to be stored in the collection is the input candidate number. This number must be less than or equal to the total number of candidates.

**GeocodeGetErrorTextV()**

**Purpose**

Given a valid MapMarker error code, this method returns a string giving a short text description of the error.

**Syntax**

```
GeocodeGetErrorTextV(errorCode As Integer,
                      errorString As VARIANT) As Boolean
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorString</td>
<td>[Output VARIANT] Returns error text.</td>
</tr>
</tbody>
</table>

**Returns**

TRUE if successful, FALSE if not.

**GeocodeGetHwyExitCandidateV()**

**Purpose**

Given a zero-based index number, this method returns highway name, exit number, and/or exit name, as well as the ZIP Code, city, and coordinates of the exit location. This method cannot be used until its corresponding call, GeocodeHwyExitV(), has been invoked. Using the count from that function, the individual candidates can be retrieved from GeocodeGetHwyExitCandidateV(). The first candidate is 0.
Syntax

GeocodeGetHwyExitCandidateV(
    index as Integer,
    HwyTypePre as VARIANT,
    HwyName as VARIANT,
    HwyTypeSuf as VARIANT,
    HwyDir as VARIANT,
    ExtNum as VARIANT,
    ExitName as VARIANT,
    City as VARIANT,
    State as VARIANT,
    Zipcode as VARIANT,
    Longitude as VARIANT,
    Latitude as VARIANT) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>[Input Integer] Zero-based index of candidates to retrieve.</td>
</tr>
<tr>
<td>HwyTypePre</td>
<td>[Output VARIANT] Pre-type of the matched highway.</td>
</tr>
<tr>
<td>HwyName</td>
<td>[Output VARIANT] Name of the matched highway.</td>
</tr>
<tr>
<td>HwyTypeSuf</td>
<td>[Output VARIANT] Post-type of the matched highway.</td>
</tr>
<tr>
<td>HwyDir</td>
<td>[Output VARIANT] The matched direction.</td>
</tr>
<tr>
<td>ExtNum</td>
<td>[Output VARIANT] The matched exit number.</td>
</tr>
<tr>
<td>ExitName</td>
<td>[Output VARIANT] The matched exit name.</td>
</tr>
<tr>
<td>City</td>
<td>[Output VARIANT] The returned city name.</td>
</tr>
<tr>
<td>State</td>
<td>[Output VARIANT] The returned state name.</td>
</tr>
<tr>
<td>Longitude</td>
<td>[Output VARIANT] The X coordinate of the returned point.</td>
</tr>
<tr>
<td>Latitude</td>
<td>[Output VARIANT] The Y coordinate of the returned point.</td>
</tr>
</tbody>
</table>

Returns

This call returns TRUE if successful, FALSE if not.
GeocodeGetServerVersionV()

Purpose
This method returns the version of the server.

Syntax
GeocodeGetServerVersionV(pVersionNum As VARIANT) As Boolean

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVersionNum</td>
<td>[Output VARIANT] MapMarker version as a whole number (i.e., 9)</td>
</tr>
</tbody>
</table>

Returns
TRUE if successful, FALSE if not.

GeocodeGetStatesFoundV()

Purpose
This method returns a list of states in the Address Dictionary path.

Syntax
GeocodeGetStatesFoundV(pstateList As VARIANT) As Boolean

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns
TRUE if successful, FALSE if not.

GeocodeGetStatesLicensedV()

Purpose
This method returns a list of licensed states.

Syntax
GeocodeGetStatesLicensedV(pstateList As VARIANT) As Boolean

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns
TRUE if successful, FALSE if not.
GeocodeHwyExitV()

Purpose
This method uses input criteria of hwyexit and state, and returns a count of the number of Hwy records that matched the input criteria. A subsequent call to GeocodeGetHwyExitCandidateV() retrieves the matched candidates.

Syntax
GeocodeHwyExitV(Hwyexit as String, State as String count as VARIANT)As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hwyexit</td>
<td>[Input String] The highway and exit information in the form: &lt;highway name&gt; &lt;highway directional&gt; EXIT&lt;exit number&gt;&lt;exit number suffix&gt;TO +&lt;exit name&gt;. Where: highway name is the name of the highway name including type prefix or type suffix (e.g., Highway 8, I-90, or Pennsylvania Turnpike. highway directional is the direction of the highway, full or abbreviated (e.g., E or East). Optional. EXIT keyword is a case-insensitive keyword used to separate the highway information from the exit information. Required. exit number is the actual number of the exit. This is optional if there is an exit name in the input. exit number suffix is any non-numerical exit suffix (e.g., Exit 2E). Optional, not used by MapMarker. TO keyword is a case-insensitive keyword used to separate the exit name and exit number. Optional. exit name is the name of the exit (e.g., State Hwy 5). Optional, MapMarker will match only to exact names when there is no exit number in the input or there is no match to the exit number in the input. Sample input: I-87 EXIT 2E TO State Hwy 5 I-87 EXIT State Hwy 5 I-87 EXIT 2E I-87 EXIT 2 I-87 EXIT 2E State Hwy. This input is used by the engine to look up the specified highway and exit in the highway exit data table (hwyexits.dbf in the Address Dictionary.</td>
</tr>
<tr>
<td>State</td>
<td>[Input String] Contains the state name, either in full or abbreviated.</td>
</tr>
<tr>
<td>count</td>
<td>[Output VARIANT] The total number of highway exit candidates that matched</td>
</tr>
</tbody>
</table>

Returns
This call returns TRUE if successful, FALSE if not.
**GeocodeIsCandidateMultiUnitV()**

**Purpose**
This method determines whether a candidate address specified by a zero-based index contains multiple units.

**Syntax**
```
GeocodeIsCandidateMultiUnitV(geocodeHandle As long,
index As Integer,
IsMultiUnit As VARIANT) As Boolean
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] The handle returned from the Geocode call used.</td>
</tr>
<tr>
<td>index</td>
<td>[Input] Index value of the candidate to examine. This index is zero-based.</td>
</tr>
<tr>
<td>IsMultiUnit</td>
<td>[Output VARIANT] True/False value indicating whether the candidate address has multiple units.</td>
</tr>
</tbody>
</table>

When using the `GeocodeIsCandidateMultiUnitV()` function with the CAddressList, keep in mind that the index for `GeocodeIsCandidateMultiUnitV()` is zero-based, and the index for CAddressList is one-based.

**Returns**
TRUE if successful; FALSE if not.

---

**GeocodePostalCentroidV()**

**Purpose**
This call can be used to geocode a ZIP Code or ZIP+4 centroid.

**Syntax**
```
GeocodePostalCentroidV(Zip As String,
Plus4 As String,
Longitude As VARIANT,
Latitude As VARIANT,
Precision As VARIANT,
ResultCode As VARIANT,
pSerialNumber As String) As Boolean
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zip</td>
<td>[Input String] ZIP Code address component</td>
</tr>
<tr>
<td>Plus4</td>
<td>[Input String] ZIP Code add-on address component</td>
</tr>
<tr>
<td>Longitude</td>
<td>[Output VARIANT] Returns the longitude value.</td>
</tr>
</tbody>
</table>
Returns
TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine what error occurred. See Appendix H: MapMarker GeoEngine Error Codes, for a complete list of errors.

GetCandidateAtV()

Purpose
This call returns the text associated with a specific candidate in the Match Candidates list box in the OCX.

Syntax
GetCandidateAtV(Index As Integer) As VARIANT

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The string of tab-delimited text associated with the specified candidate.
GetCandidateCensusBlockIDAtV()

Purpose
This call returns the Census Block ID of an address in the Match Candidates list box in the OCX. The Census Block ID is a code of up to 15 digits and characters that describes the smallest of the U.S. Census Bureau census units.

Syntax
GetCandidateCensusBlockIDAtV(Index As Integer) As VARIANT

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The Census Block ID of the specified candidate.

GetCandidateCityAtV()

Purpose
This call returns the city portion of an address in the Match Candidates list box in the OCX.

Syntax
GetCandidateCityAtV(Index As Integer) As VARIANT

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The city of the specified candidate.

GetCandidateFirmAtV()

Purpose
This call returns the firm portion of an address to the Match Candidates list box in the OCX.

Syntax
GetCandidateFirmAtV(Index As Integer) As VARIANT
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The firm of the specified candidate.

**GetCandidateLatitudeAtV()**

**Purpose**
This call returns the latitude of an address in the Match Candidates list box in the OCX.

**Syntax**
GetCandidateLatitudeAtV(Index As Integer) As VARIANT

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The latitude of the specified candidate.

**GetCandidateLongitudeAtV()**

**Purpose**
This call returns the longitude of an address in the Match Candidates list box in the OCX.

**Syntax**
GetCandidateLongitudeAtV(Index As Integer) As VARIANT

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The longitude of the specified candidate.
GetCandidatePlus4AtV()

Purpose
This call returns the ZIP Add-on portion of an address in the Match Candidates list box in the OCX.

Syntax
GetCandidatePlus4AtV(Index As Integer) As VARIANT

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The ZIP Add-on (Plus 4) of the specified candidate.

GetCandidatePrecisionAtV()

Purpose
This call returns the precision portion of an address in the Match Candidates list box in the OCX. The precision defines the type of match: street level, shape path, intersection, Point ZIP, or ZIP centroid (either ZIP+4, ZIP+2 or ZIP Code).

Syntax
GetCandidatePrecisionAtV(Index As Integer) As VARIANT

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The precision of the specified candidate.

GetCandidateResultCodeAtV()

Purpose
Returns the result code for an address in the Match Candidates list box in the OCX. It represents the type of match (single, multiple, or ZIP centroid) and how precisely the GeoEngine matched to the address components (house number, street name, prefix, street type, city name, ZIP Code, and whether it matched to the Address Dictionary or user defined dictionary.)

Syntax
GetCandidateResultCodeAtV(Index As Integer) As VARIANT
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

### Returns

The result code of the specified candidate.

### GetCandidateStateAtV()

**Purpose**

This call returns the state portion of an address in the Match Candidates list box in the OCX.

**Syntax**

```plaintext```
GetCandidateStateAtV(Index As Integer) As VARIANT
```plaintext```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

### Returns

The state of the specified candidate.

### GetCandidateStreetAtV()

**Purpose**

This call returns the street portion of an address in the Match Candidates list box in the OCX.

**Syntax**

```plaintext```
GetCandidateStreetAtV(Index As Integer) As VARIANT
```plaintext```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

### Returns

The street address of the specified candidate.
GetCandidateZIPAtV()

Purpose
This call returns the ZIP Code portion of an address in the Match Candidates list box in the OCX

Syntax
GetCandidateZIPAtV(Index As Integer) As VARIANT

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.</td>
</tr>
</tbody>
</table>

Returns
The ZIP Code of the specified candidate.
ASP Sample Application

Use the code within SampleApp.asp on page 224 and SampleAppForm.asp on page 227 to create a sample ASP geocoding application and input form.

SampleApp.asp

```vbscript
<%@ Language=VBScript%>
<%option explicit%>

<%
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'sampleApp.asp
'
'This page takes user input from sampleAppForm.asp and is
called via the "post" method
'
'The output is written to a web page.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'DECLARATIONS

'Geocoding Results
Dim geoHandle
Dim geoStatus
Dim numCandidates
Dim numCloseCandidates
Dim retval
Dim errorCode
Dim errorMessage
Dim AdrList 'Container to store candidate address structures using GeocodeGetCandidatesV()

Dim Adr 'Candidate address structure
Dim sess_g_objMM
Dim i
Dim line1
Dim line2
Dim line3
Dim line4
```
'Main Program
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
For Each Adr in AdrList

    line2 = Adr.Latitude & " / " & Adr.Longitude
    line3 = "Precision = " & Adr.Precision
    line4 = Adr.ResultCode

    Response.Write(line1 & "<BR>")
    Response.Write(line2 & "<BR>")
    Response.Write(line3 & "<BR>")
    Response.Write(line4 & "<BR>")

Next
End If

'Free Memory
If geoHandle <> 0 Then
    If Not sess_g_objMM.GeocodeFreeSetV(geoHandle) Then
        Response.Write("ERROR FREEING MEMORY!")
    End If
End If
End If

Response.End
Session.Abandon

%>
SampleAppForm.asp

<%@ Language=VBScript%>
<%option explicit%>

<html>
<head></head>
<body bgcolor="lightblue">

<!-- SampleAppForm.asp
This sample form collects user input for use with SampleApp.asp. -->

<form name="form1" action="SampleApp.asp" method="post">

<fieldset>
  <legend>
    Preferences
  </legend>
  <pre><input type="checkbox" name="match_intersections"> Match Intersections</pre>
</fieldset>

<fieldset>
  <legend>
    GeocodeAddressV
  </legend>
  <pre>Street:  <input type="text" name="street1"><BR>
    City:  <input type="text" name="city"> State:  <input type="text" name="state" size=3> Zip:  <input type="text" size=8 name="zip1"> <BR>
    <input type="submit" value="Geocode" name="_submit"></pre>
</fieldset>

<fieldset>
  <legend>
    GeocodeAddressLastlineV
  </legend>
  <pre>Street: <input type="text" name="street2"> City,State,Zip: <input type="text" name="lastline" size=25><BR>
    <input type="submit" value="GeocodeLastline" name="_submit"></pre>
</fieldset>

<fieldset>
  <legend>
    GeocodePostalCentroidV
  </legend>
  <pre>Zip: <input type="text" size=6 name="zip2"> Zip+4: <input type="text" size=4 name="zip4"><BR>
    <input type="submit" value="GeocodeCentroid" name="_submit"></pre>
</fieldset>

</form>

</body>
</html>
<fieldset>
  <legend>
    Serial Number
  </legend>
  <p>This is an optional argument for each of the above geocoding options. It is used for OEM applications where the server requires a serial # for geocoding. If this option is used, an OEM license file must be created and stored in the data directory.</p>
  <pre>
    Serial#:<input type="text" name="serial">
  </pre>
</fieldset>
C programmers using Remote Procedure Call (RPC) now have a tool to develop custom geocoding applications that call the MapMarker Server, providing nationwide geocoding services for virtually any Windows desktop.

The MapMarker RPC API is a collection of methods that allows you to connect to the MapMarker Server and geocode records from a client. This chapter describes the API and assists you in developing a custom geocoding application. We assume that you are familiar with RPC and how to connect to remote servers.

In this chapter:

- MapMarker Server RPC Interface .......................................................... 230
- Using the RPC API to Build a Remote Geocoding Application ............ 230
- RPC Server Error Codes ................................................................. 230
- API Function Calls ........................................................................ 231
MapMarker Server runs as a service and can be stopped and started using the Service Control Manager. It also runs as a console application in Windows 2000, 2003, or XP. See Chapter 9: Client/Server Geocoding.

The Server grabs a local RPC connection and a TCP/IP socket with a dynamically assigned endpoint. It registers the endpoints in the machine’s endpoint registry. A client can connect to the process locally using ncalrpc and remotely using a partially bound ncacn_ip_tcp handle.

The server uses RPC context_handles to store local information obtained during a call to GeocodeAddress(). This enables the candidates and other information to be retrieved by the client after the geocoding is complete. The Geocoding Engine is not thread safe, and as a result, the server prevents multiple threads from using it by guarding the GEOCODEHANDLE with a mutex. The time involved in geocoding a single address is very small, so threads should never have to wait long.

Using the RPC API to Build a Remote Geocoding Application

The following files are included in the RPC toolkit:

- mmdist.h (prototypes for RPC calls)
- mmdist_c.c (client-side RPC stubs)
- mmdist.idl (interface definition file)
- rpcrt4.lib (Microsoft run time library for RPC).

The first three files are found in the \mapmarker\geoeng\rpc subdirectory after installation.

When compiling your client application, you need to include the mm_dist.h and mm_distc.c files. Your application also needs to link against the Microsoft run time library for RPC (rpcrt4.lib).

RPC Server Error Codes

The following table provides an explanation of the RPC errors you may encounter.

<table>
<thead>
<tr>
<th>Error #</th>
<th>Error</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1702</td>
<td>RPC_S_INVALID_BINDING</td>
<td>Server cannot be found.</td>
</tr>
<tr>
<td>1722</td>
<td>RPC_S_SERVER_UNAVAILABLE</td>
<td>The RPC server is unavailable.</td>
</tr>
<tr>
<td>1723</td>
<td>RPC_S_SERVER_TOO_BUSY</td>
<td>The RPC server is too busy to complete the operation.</td>
</tr>
</tbody>
</table>
API Function Calls

- **DpvGeocodeAddress()** ................................................................. 232
  Turns DPV mode on, and then attempts to geocode an address and build a list of candidates.

- **DpvGeocodeAddressLastLine()** .................................................. 234
  Turns DPV mode on, and then attempts to geocode an address and build a list of candidates.

- **DpvGeocodeAddressLastLineWithSerial()** .................................... 235
  Turns DPV mode on, and then attempts to geocode an address and build a list of candidates using an unparsed last line instead of city, state, and ZIP fields.

- **DpvGeocodeAddressWithSerial()** ............................................... 237
  Turns DPV mode on, and then attempts to geocode an address and build a list of candidates in server applications that are initialized with a serial number.

- **GeocodeAddress()** .................................................................. 239
  Geocodes an address and builds a list of candidates.

- **GeocodeAddressEx()** ................................................................. 240
  Geocodes an address and builds a list of candidates. Enables street2 information to be added as a separate field.

- **GeocodeAddressLastLine()** ....................................................... 242
  Geocodes an address and builds a list of candidates accepting a last line instead of City, State, and ZIP.

- **GeocodeAddressLastLineEx()** ................................................... 243
  Geocodes an address and builds a list of candidates accepting a last line instead of City, State, and ZIP. Enables street2 information to be added as a separate field.

- **GeocodeAddressLastLineWithSerial()** ...................................... 245
  Geocodes an address and builds a list of candidates using a server that is initialized with a serial number accepting a last line instead of City, State, and ZIP.

- **GeocodeAddressLastLineWithSerialEx()** .................................. 246
  Geocodes an address and builds a list of candidates using a server that is initialized with a serial number accepting a last line instead of City, State, and ZIP. Enables street2 information to be passed in as a separate field.

- **GeocodeAddressWithSerial()** ................................................... 248
  Geocodes an address and builds a list of candidates using a server that is initialized with a serial number.

- **GeocodeAddressWithSerialEx()** ................................................. 249
  Geocodes an address and builds a list of candidates using a server that is initialized with a serial number. Enables street2 information to be passed in as a separate field.

- **GeocodeCheckDbAvailability()** ............................................... 251
  Checks to see which databases are available for geocoding.

- **GeocodeFreeSet()** ................................................................... 251
  Frees all the server side information.

- **GeocodeGetCandidate()** ......................................................... 252
  Gets all the needed candidate information.

- **GeocodeGetCandidateEx()** ...................................................... 253
  Gets all needed candidate information, plus the record type, delivery point, carrier route, check digit, primary street, tabbedAddress, lacs, pmb, pmb_range, and addressType. These parameters may not have values.
- **GeocodeGetCandidateCoords()**  \(\ldots\) 255
  Retrieves geocode point and precision for a specific candidate.

- **GeocodeGetCandidateWithDpv()**  \(\ldots\) 255
  This call is used to obtain necessary candidate information from the
  `DpvGeocodeAddress()` call.

- **GeocodeGetErrorText()**  \(\ldots\) 258
  Returns a string giving a short text description of the error.

- **GeocodeGetHwyExitCandidate()**  \(\ldots\) 259
  Geocodes a highway exit following call to `GeocodeHwyExit()`.

- **GeocodeGetServerVersion()**  \(\ldots\) 260
  Returns the server version number.

- **GeocodeGetStatesFound()**  \(\ldots\) 260
  Returns a list of states found in the Address Dictionary Path.

- **GeocodeGetStatesLicensed()**  \(\ldots\) 260
  Returns a list of states currently licensed.

- **GeocodeHwyExit()**  \(\ldots\) 261
  This method returns a count of the number of highway exit records based on input
  criteria of highway exit and state.

- **GeocodeIsCandidateMultiUnit()**  \(\ldots\) 262
  Determines whether a candidate address specified by a zero-based index contains
  multiple units.

- **GeocodePostalCentroid()**  \(\ldots\) 263
  Gets a ZIP Code or ZIP+4 centroid.

- **GeocodePostalCentroidWithSerial()**  \(\ldots\) 264
  Gets a ZIP Code or ZIP+4 centroid using a server that is initialized with a serial
  number.

- **GeocodeShutdown()**  \(\ldots\) 264
  Shuts down the MapMarker Server.

---

**DpvGeocodeAddress()**

**Purpose**

This call turns DPV mode on, and then attempts to geocode an address and build a list of
candidates. It also enables street2 information to be passed in as a separate field. Note that CASS
mode must be turned on before DPV mode can be activated. When using any of the DPV
geocoding functions, you must use `GeocodeGetCandidateWithDpv` to retrieve the DPV
information.
**Syntax**

```c
long DpvGeocodeAddressEx(Geocode_handle*geocodeHandle,
char *pFirm,
char *pStreet,
char *Street2,
char *pCity,
char *pState,
char *pZIP,
short relaxFlags,
double linearInset,
double perpendicularSetback,
short distUnit,
short relaxDistance,
short *Status,
short *NumCandidates,
short *NumCloseMatch)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] Context handle needed to reference internal items. A handle is retrieved by the call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet2</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pCity</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pState</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pZIP</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match. These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] Distance from street corner. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] Distance from street. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Unit used for linearInset and perpendicularSetback. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Distance to relax finance area search. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>Status</td>
<td>[Output] The match status of the geocode. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>NumCandidates</td>
<td>[Output] The total number of candidates.</td>
</tr>
<tr>
<td>NumCloseMatch</td>
<td>[Output] Number of close match candidates.</td>
</tr>
</tbody>
</table>
Returns
The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

DpvGeocodeAddressLastLine()

Purpose
This call turns DPV mode on, and then attempts to geocode an address and build a list of candidates. DpvGeocodeAddressLastLine() does the same things as DpvGeocodeAddress() except that it accepts a last line instead of city, state, and ZIP fields. Note that CASS mode must be turned on before DPV mode can be activated. When using any of the DPV geocoding functions, you must use GeocodeGetCandidateWithDpv to retrieve the DPV information.

Syntax

```c
long DpvGeocodeAddressLastLine(GEOCODE_HANDLE *geocodeHandle,
    char *pFirm,
    char *pStreet,
    char *pStreet2,
    char *pLastLine,
    short relaxFlags,
    double linearInset,
    double perpendicularSetback,
    short distUnit,
    short *pStatus,
    short *pNumCandidates,
    short *pNumCloseCandidates)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet2</td>
<td>[Input] Secondary address.</td>
</tr>
<tr>
<td>pLastLine</td>
<td>[Input] The (unparsed) city, state, and ZIP information.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match. These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
</tbody>
</table>
Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

DpvGeocodeAddressLastLineWithSerial()

Purpose

This call turns DPV mode on, and then attempts to geocode an address and build a list of candidates using an unparsed last line instead of city, state, and ZIP fields. This call is used in server applications where the server is initialized with a serial number. This call also enables you to pass in street2 information as a separate field. Note that CASS mode must be turned on before DPV mode can be activated.
Syntax

```c
long DpvGeocodeAddressLastLineWithSerialEx(Geocode_Handle *geocodeHandle,
                                           char *pFirm,
                                           char *pStreet,
                                           char *pStreet2,
                                           char *pLastline,
                                           short relaxFlags,
                                           double linearInset,
                                           double perpendicularSetback,
                                           short distUnit,
                                           short relaxDistance,
                                           short *status,
                                           short *numCandidates,
                                           short *numCloseMatch,
                                           char *pSerialNumber)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] Context handle needed to reference internal items. A handle is retrieved by the call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Firm name.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Street name.</td>
</tr>
<tr>
<td>pStreet2</td>
<td>[Input] Secondary address.</td>
</tr>
<tr>
<td>pLastline</td>
<td>[Input] The (unparsed) city, state, and ZIP information.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match (i.e., in the Geocode dialog). These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] Distance from street corner. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] Distance from street Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Distance to relax finance area search. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
<tr>
<td>status</td>
<td>[Output] The match status of the geocode. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseMatch</td>
<td>[Output] The number of close match candidates.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Input] The serial number for OEM license usage.</td>
</tr>
</tbody>
</table>
Returns

The call returns 0 if successful; otherwise one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

DpvGeocodeAddressWithSerial()

Purpose

This call turns DPV mode on, and then attempts to geocode an address and build a list of candidates in server applications that are initialized with a serial number. This call also enables you to pass street2 information in as a separate field. Note that CASS mode must be turned on before DPV mode can be activated.

Syntax

```c
long DpvGeocodeAddressWithSerialEx(Geocode_Handle*geocodeHandle,
char *pFirm,
char *pStreet,
char *pStreet2,
char *pCity,
char *State,
char *pZip,
short relaxFlags,
double linearInset,
double perpendicularSetback,
short distUnit,
short relaxDistance,
short *status,
short *numCandidates,
short *numCloseMatch,
char *pSerialNumber)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] Context handle needed to reference internal items. A handle is retrieved by the call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet2</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pCity</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pState</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pZip</td>
<td>[Input] Address component to be matched.</td>
</tr>
</tbody>
</table>
Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match (i.e., in the Geocode dialog). These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] Distance from street. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] Distance from street. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Possible values are found in geostd.h. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Distance to relax finance area search. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>status</td>
<td>[Output] the match status of the geocode. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseMatch</td>
<td>[Output] The number of close match candidates.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Input] The serial number for OEM license usage.</td>
</tr>
</tbody>
</table>
GeocodeAddress()

Purpose
This is the main call for the RPC API. It attempts to geocode an address and build a list of candidates.

Syntax

```c
long GeocodeAddress(GEOCODE_HANDLE *geocodeHandle,
    char *pFirm,
    char *pStreet,
    char *pCity,
    char *pState,
    char *pZIP,
    short relaxFlags,
    double linearInset,
    double perpendicularSetback,
    short distUnit,
    short *pStatus,
    short *pNumCandidates,
    short *pNumCloseCandidates)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pCity</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pState</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pZIP</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match. These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] The distance the point lies from a street corner. Set this parameter to MMD_Default_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] The distance from the center of the street. Set this parameter to MMD_Default_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Possible values are found in geostd.h. Set this parameter to MMD_Default_CFG or -1 to use the current Registry settings.</td>
</tr>
</tbody>
</table>
relaxFlags Parameter

If relaxFlags is set to MMD_Default_CFG or -1, the server uses the geocoding preferences that are set in the Registry. You can override those settings by specifying which preferences to relax. If more than one preference is to be relaxed, the constants are added together. For example, to relax house number and street name, the relaxFlags parameter should be set to MMD_RELAX_HOUSE + MMD_RELAX_NAME, or 5.

The following is a list of flags for the relaxFlags parameter:

- MMD_RELAX_HOUSE (1)
- MMD_RELAX_POSTAL_CODE (2)
- MMD_RELAX_NAME (4)
- MMD_RELAX_FINANCE (8)
- MMD_RELAX_FINANCE_IN_STATE (16)
- MMD_USE_CASS_MODE (32)
- MMD_MATCH_INTERSECTIONS (64)
- MMD_RELAX_CITY (128)
- MMD_PREFER_UD (256)
- MMD_DEFAULT_CFG (-1)

Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then geocodeHandle is used to reference information about the geocoding operation in other calls.

GeocodeAddressEx() Purpose

This call attempts to geocode an address and build a list of candidates. It works similarly to GeocodeAddress(); however, it also enables street2 information to be passed in as a separate field.
Syntax

long GeocodeAddressEx(Geocode_handle*geocodeHandle,
  char *pFirm,
  char *pStreet,
  char *Street2,
  char *pCity,
  char *pState,
  char *pZIP,
  short relaxFlags,
  double linearInset,
  double perpendicularSetback,
  short distUnit,
  short relaxDistance,
  short *Status,
  short *NumCandidates,
  short *NumCloseMatch)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] Context handle needed to reference internal items. A handle is retrieved by the call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet2</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pCity</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pState</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pZip</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match. These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] Distance from street corner. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] Distance from street. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Unit used for linearInset and perpendicularSetback. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Distance to relax finance area search. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>status</td>
<td>[Output] The match status of the geocode. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseMatch</td>
<td>[Output] Number of close match candidates.</td>
</tr>
</tbody>
</table>
Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

GeocodeAddressLastLine()

Purpose

This call attempts to geocode an address and build a list of candidates.

GeocodeAddressLastLine() does the same things as GeocodeAddress() except that it accepts a last line instead of city, state, and ZIP fields.

Syntax

```c
long GeocodeAddressLastLine(GEOCODE_HANDLE *geocodeHandle,
                          char *pFirm,
                          char *pStreet,
                          char *pLastLine,
                          short relaxFlags,
                          double linearInset,
                          double perpendicularSetback,
                          short distUnit,
                          short relaxDistance,
                          short *pStatus,
                          short *pNumCandidates,
                          short *pNumCloseCandidates)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pLastLine</td>
<td>[Input] The (unparsed) city, state, and ZIP information.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match. These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] The distance the point lies from a street corner. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
</tbody>
</table>
Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

GeocodeAddressLastLineEx()

Purpose

This call attempts to geocode an address and build a list of candidates using an unparsed last line instead of city, state, and ZIP fields. It works similarly to GeocodeAddressLastLine(); however, it also enables you to pass in street2 information as a separate field.

Syntax

```c
long GeocodeAddressLastLineEx( Geocode_Handle *geocodeHandle, 
    char *pFirm, 
    char *pStreet, 
    char *pStreet2, 
    char *pLastline, 
    short relaxFlags, 
    double linearInset, 
    double perpendicularSetback, 
    short distUnit, 
    short relaxDistance, 
    short *status, 
    short *numCandidates, 
    short *numCloseMatch)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>perpendicularSetback</td>
<td>[Input] The distance from the center of the street. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Possible values are found in geostd.h. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Specifies the distance on a relaxed finance search. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>pStatus</td>
<td>[Output] Returns the match status of a candidate. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>pNumCandidates</td>
<td>[Output] The number of match candidates for the address.</td>
</tr>
<tr>
<td>pNumCloseCandidates</td>
<td>[Output] The number of close candidates for an address.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] Context handle needed to reference internal items. A handle is retrieved by the call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Firm name.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Street name.</td>
</tr>
<tr>
<td>pStreet2</td>
<td>[Input] Secondary address.</td>
</tr>
<tr>
<td>pLastline</td>
<td>[Input ] The (unparsed) city, state, and ZIP information.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input]. Specifies values that control the match. These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input ] Distance from street corner. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] Distance from street. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Distance to relax finance area search. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>status</td>
<td>[Output] The match status of the geocode. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseMatch</td>
<td>[Output] The number of close match candidates.</td>
</tr>
</tbody>
</table>

Returns

The call returns 0 if successful; otherwise one of the MapMarker errors codes is returned. See Appendix H: MapMarker GeoEngine Error Codes for full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.
GeocodeAddressLastLineWithSerial()

**Purpose**
This call attempts to geocode an address and build a list of candidates. This function does the same thing as GeocodeAddressWithSerial() except it accepts an unparsed last line instead of city, state, and ZIP fields.

**Syntax**
```c
long GeocodeAddressLastLineWithSerial(GEOCODE_HANDLE *geocodeHandle,
        char *pFirm,
        char *pStreet,
        char *pLastLine,
        short relaxFlags,
        double linearInset,
        double perpendicularSetback,
        short distUnit,
        short relaxDistance,
        short *pStatus,
        short *pNumCandidates,
        short *pNumCloseCandidates
        char *pSerialNumber)
```

**Parameters.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pLastLine</td>
<td>[Input] The (unparsed) city, state, and ZIP information.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match (i.e., in the Geocode dialog). These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] The distance the poinSet this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.t lies from a street corner.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] The distance from the center of the street.Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Possible values are found in geostd.h. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
</tbody>
</table>
Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

GeocodeAddressLastLineWithSerialEx()

Purpose

This call attempts to geocode an address and build a list of candidates using an unparsed last line instead of city, state, and ZIP fields. It works similarly to GeocodeAddressLastLineWithSerial(); however, it also enables you to pass in street2 information as a separate field.

Syntax

```c
long GeocodeAddressLastLineWithSerialEx(Geocode_Handle *geocodeHandle,
char *pFirm,
char *pStreet,
char *pStreet2,
char *pLastline,
short relaxFlags,
double linearInset,
double perpendicularSetback,
short distUnit,
short relaxDistance,
short *status,
short *numCandidates,
short *numCloseCandidates,
char *pSerialNumber)
```

Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relaxDistance</td>
<td>[Input] Specifies the distance on a relaxed finance seSet this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.arch.</td>
</tr>
<tr>
<td>pStatus</td>
<td>[Output] Returns the match status of a candidate. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>pNumCandidates</td>
<td>[Output] The number of match candidates for the address.</td>
</tr>
<tr>
<td>pNumCloseCandidates</td>
<td>[Output] The number of close candidates for an address.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Input] Serial number for OEM licenses.</td>
</tr>
</tbody>
</table>
MapMarker Plus 10.0

User Guide
Chapter 13: MapMarker Server RPC API

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] Context handle needed to reference internal items. A handle is retrieved by the call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Firm name.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Street name.</td>
</tr>
<tr>
<td>pStreet2</td>
<td>[Input] Secondary address.</td>
</tr>
<tr>
<td>pLastline</td>
<td>[Input] The (unparsed) city, state, and ZIP information.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match (i.e., in the Geocode dialog). These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] Distance from street corner. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] Distance from Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings. street.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Distance to relax finance area search. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
<tr>
<td>status</td>
<td>[Output] The match status of the geocode. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseMatch</td>
<td>[Output] The number of close match candidates.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Input] The serial number for OEM license usage.</td>
</tr>
</tbody>
</table>

Returns

The call returns 0 if successful; otherwise one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.
GeocodeAddressWithSerial()

Purpose
This call attempts to geocode an address and build a list of candidates. Use this function instead of GeocodeAddress() in server applications that are initialized with a serial number.

Syntax

```c
long GeocodeAddressWithSerial (GEOCODE_HANDLE *geocodeHandle,
    char *pFirm,
    char *pStreet,
    char *pCity,
    char *pState,
    char *pZIP,
    short relaxFlags,
    double linearInset,
    double perpendicularSetback,
    short distUnit,
    short relaxDistance,
    short *pStatus,
    short *pNumCandidates,
    short *pNumCloseCandidates,
    char *pSerialNumber)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pCity</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pState</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pZIP</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match (i.e., in the Geocode dialog). These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] The distance the point lies from a street corner. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] The distance from the center of the street. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry Settings.</td>
</tr>
</tbody>
</table>
Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

### GeocodeAddressWithSerialEx()

**Purpose**

This call attempts to geocode an address and build a list of candidates in server applications that are initialized with a serial number. It works similarly to GeocodeAddressWithSerial(); however, it also enables you to pass street2 information in as a separate field.

**Syntax**

```c
long GeocodeAddressWithSerialEx(Geocode_Handle*geocodeHandle, 
    char *pFirm, 
    char *pStreet, 
    char *pStreet2, 
    char *pCity, 
    char *State, 
    char *pZip, 
    short relaxFlags, 
    double linearInset, 
    double perpendicularSetback, 
    short distUnit, 
    short relaxDistance, 
    short *status, 
    short *numCandidates, 
    short *numCloseMatch, 
    char *pSerialNumber)```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Specifies the distance on a relaxed finance search. Set this parameter to MMD_DEFAULT_CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>pStatus</td>
<td>[Output] Returns the match status of a candidate. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>pNumCandidates</td>
<td>[Output] The number of match candidates for the address.</td>
</tr>
<tr>
<td>pNumCloseCandidates</td>
<td>[Output] The number of close candidates for an address.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Input] Serial number for OEM licenses.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output] Context handle needed to reference internal items. A handle is retrieved by the call.</td>
</tr>
<tr>
<td>pFirm</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pStreet2</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pCity</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pState</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>pZip</td>
<td>[Input] Address component to be matched.</td>
</tr>
<tr>
<td>relaxFlags</td>
<td>[Input] Specifies values that control the match (i.e., in the Geocode dialog). These are used to override the server settings. Values are found in mmdist.h. See relaxFlags Parameter on page 240 for more information.</td>
</tr>
<tr>
<td>linearInset</td>
<td>[Input] Distance from street. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>[Input] Distance from street. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>distUnit</td>
<td>[Input] Specifies the distance unit used for linearInset and perpendicularSetback. Possible values are found in geostd.h. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>relaxDistance</td>
<td>[Input] Distance to relax finance area search. Set this parameter to MMD_DEFAULT CFG or -1 to use the current Registry settings.</td>
</tr>
<tr>
<td>status</td>
<td>[Output] the match status of the geocode. Possible values are found in geostd.h.</td>
</tr>
<tr>
<td>numCandidates</td>
<td>[Output] The total number of candidates.</td>
</tr>
<tr>
<td>numCloseMatch</td>
<td>[Output] The number of close match candidates.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Input] The serial number for OEM license usage.</td>
</tr>
</tbody>
</table>

Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.
GeocodeCheckDbAvailability()

Purpose
This call checks to see what databases are available for geocoding: Street, ZIP centroids, User Dictionary, or some combination.

Syntax
long GeocodeCheckDbAvailability (short *dbTypes);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns
The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

GeocodeFreeSet()

Purpose
Whenever GeocodeAddress() is invoked, a memory structure is allocated, the structure is populated with a list of possible candidates and a context handle to that structure is returned to the caller. GeocodeFreeSet() releases the memory structure allocated to hold the candidate list. Once the information is freed, the handle cannot be used again, unless the same value is returned by a later call to GeocodeAddress(). MapMarker can allocate a maximum of 1,024 of these structures. To avoid running out of memory or blocking the MapMarker Server, call GeocodeFreeSet() as soon as you have finished evaluating a candidate list.

Syntax
short GeocodeFreeSet(GEOCODE_HANDLE*geocodeHandle)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] The context handle that references the geocoding memory structure which is to be released.</td>
</tr>
</tbody>
</table>

Returns
The call returns 0 if successful, and 1 if failed.
GeocodeGetCandidate()

Purpose
This call is used to get all the needed candidate information from the GeocodeAddress() call. If the match status was SINGLE_MATCH, then this call with a candidateNumber of 0 would return the candidate information for the matched address. The candidate list is ordered with close matches at the top.

Syntax

```c
long GeocodeGetCandidate (GEOCODE_HANDLE geocodeHandle,
                        short candidateNumber,
                        char *firm[MMD_FIRM_LEN],
                        char *streetAddress[MMD_STREET_LEN],
                        char *city[MMD_CITY_LEN],
                        char *state[MMD_STATE_LEN],
                        char *zip[MMD_ZIP_LEN],
                        char *plus4[MMD_PLUS4_LEN],
                        double *longitude,
                        double *latitude,
                        short *precision,
                        char *resultcode[MMD_RESULT_CODE],
                        char *censusBlockID[MMD_CENSUS_BLOCK_LEN])
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>candidateNumber</td>
<td>[Input] The index of the desired candidate. The first candidate will have an index of 0.</td>
</tr>
<tr>
<td>firm</td>
<td>[Output] Candidate address component.</td>
</tr>
<tr>
<td>streetAddress</td>
<td>[Output] Candidate address component.</td>
</tr>
<tr>
<td>city</td>
<td>[Output] Candidate address component.</td>
</tr>
<tr>
<td>state</td>
<td>[Output] Candidate address component.</td>
</tr>
<tr>
<td>zip</td>
<td>[Output] Candidate address component.</td>
</tr>
<tr>
<td>plus4</td>
<td>[Output] Candidate address component.</td>
</tr>
<tr>
<td>longitude</td>
<td>[Output] Longitude of geocoded point.</td>
</tr>
<tr>
<td>latitude</td>
<td>[Output] Latitude of geocoded point.</td>
</tr>
<tr>
<td>resultcode</td>
<td>[Output] Result code of the matched address.</td>
</tr>
<tr>
<td>censusBlockID</td>
<td>[Output] The Census ID of the candidate. (See Appendix E: Frequently Asked Questions, for more information about Census IDs.)</td>
</tr>
</tbody>
</table>
Returns

This function returns 0 on success, or one of the following values:

- 1 – The geocode handle was invalid or already freed.
- 2 – The candidateNumber is out of range.

Note: Following a successful geocode, you must free the candidates using the GeocodeFreeSet() call. Otherwise it eventually stops the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle parameter is used to reference information about the geocoding operation in other calls.

GeocodeGetCandidateEx()

Purpose

This call is used to obtain necessary candidate information from the GeocodeAddress() call. If the status was SINGLE_MATCH, then this call with a candidateNumber of 0 would return the match and coordinate information for the matched address.

This call works similarly to GeocodeGetCandidate(); however, it also returns the record type, delivery point, carrier route, check digit, and primary street. These parameters may not have values.

Syntax

```c
long GeocodeCandidateEx(Geocode_Handle geocodeHandle, 
                      short candidateNumber, 
                      char firm[MMD_FIRM_LEN], 
                      char streetAddress[MMD_STREET_LEN], 
                      char city[MMD_CITY_LEN], 
                      char state[MMD_STATE_LEN], 
                      char zip[MMD_ZIP_LEN], 
                      char plus4[MMD_PLUS4_LEN], 
                      char *recordType, 
                      char deliveryPoint[MMD_DELIVERY_POINT_LEN], 
                      char carrierRoute[MMD_CARRIER_ROUTE_LEN], 
                      char checkDigit[MMD_CHECK_DIGIT_LEN], 
                      char primaryStreet[MMD_PRIMARYSTREET_LEN], 
                      char tabbedAddress[MMD_STREET_LEN], 
                      char lacs[MMD_LACS_LEN], 
                      char pmb[MMD_PMB_LEN], 
                      char pmb_range[MMD_PMB_RANGE_LEN] 
                      short *addressType 
                      double *longitude, 
                      double *latitude, 
                      short *precision 
                      char resultCode[MMD_RESULT_CODE], 
                      char censusBlockId[MMD_CENSUS_BLOCK_LEN])
```
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>candidateNumber</td>
<td>[Input] The index of the desired candidate. The first candidate will have an index of 0.</td>
</tr>
<tr>
<td>firm</td>
<td>[Output] Candidate firm</td>
</tr>
<tr>
<td>streetAddress</td>
<td>[Output] Candidate street address</td>
</tr>
<tr>
<td>city</td>
<td>[Output] Candidate city</td>
</tr>
<tr>
<td>state</td>
<td>[Output] Candidate state</td>
</tr>
<tr>
<td>zip</td>
<td>[Output] Candidate ZIP code</td>
</tr>
<tr>
<td>recordType</td>
<td>[Output] Record type for candidate.</td>
</tr>
<tr>
<td>deliveryPoint</td>
<td>[Output] Delivery Point for candidate address.</td>
</tr>
<tr>
<td>carrierRoute</td>
<td>[Output] Carrier Route for candidate address.</td>
</tr>
<tr>
<td>checkDigit</td>
<td>[Output] Check Digit for candidate address.</td>
</tr>
<tr>
<td>primaryStreet</td>
<td>[Output] Primary street for candidate address.</td>
</tr>
<tr>
<td>tabbedAddress</td>
<td>[Output] Candidate tabbed address.</td>
</tr>
<tr>
<td>lacs</td>
<td>[Output] Candidate Lacs.</td>
</tr>
<tr>
<td>pmb</td>
<td>[Output] Candidate PMB.</td>
</tr>
<tr>
<td>pmb_range</td>
<td>[Output] Candidate PMB range/number.</td>
</tr>
<tr>
<td>addressType</td>
<td>[Output] Candidate MapMarker address type.</td>
</tr>
<tr>
<td>longitude</td>
<td>[Output] Longitude of geocoded point.</td>
</tr>
<tr>
<td>latitude</td>
<td>[Output] Latitude of geocoded point.</td>
</tr>
<tr>
<td>precision</td>
<td>[Output] Code representing the precision of the geocoded point.</td>
</tr>
<tr>
<td>resultcode</td>
<td>[Output] The candidate result code.</td>
</tr>
<tr>
<td>censusBlockId</td>
<td>[Output] The census id of the candidate.</td>
</tr>
</tbody>
</table>

## Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See **Appendix H: MapMarker GeoEngine Error Codes** for a full listing of error codes.
GeocodeGetCandidateCoords()

Purpose
This call retrieves geocode point and precision for a specific candidate in the candidate list.

Syntax

```
GeocodeGetCandidateCoords(pRPC_GEOCODE_HANDLE geocodeHandle,
    short candidateNumber,
    double *longitude,
    double *latitude,
    short *precision)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>candidateNumber</td>
<td>[Input] The index of the desired candidate.</td>
</tr>
<tr>
<td>longitude</td>
<td>[Output] Longitude of geocoded point.</td>
</tr>
<tr>
<td>latitude</td>
<td>[Output] Latitude of geocoded point.</td>
</tr>
<tr>
<td>precision</td>
<td>[Output] Precision of the coordinate. See geostd.h for a complete list of precision returns.</td>
</tr>
</tbody>
</table>

Returns )

This function returns 0 on success, or one of the following values:

- 1 – The geocode handle was invalid or already freed.
- 2 – The candidateNumber is out of range.

GeocodeGetCandidateWithDpv()

Purpose
This call is used to obtain necessary candidate information from the DpvGeocodeAddress() call. If the status was SINGLE_MATCH, then this call with a candidateNumber of 0 would return the match and coordinate information for the matched address.

This call returns everything that GeocodeGetCandidateEx() returns, plus the DPV information.
Syntax

```c
long GeocodeCandidateWithDpv(Geocode_Handle geocodeHandle,
   short candidateNumber,
   char firm[MMD_FIRM_LEN],
   char streetAddress[MMD_STREET_LEN],
   char city[MMD_CITY_LEN],
   char state[MMD_STATE_LEN],
   char zip[MMD_ZIP_LEN],
   char plus[4][MMD_PLUS4_LEN],
   char *recordType,
   char deliveryPoint[MMD_DELIVERY_POINT_LEN],
   char carrierRoute[MMD_CARRIER_ROUTE_LEN],
   char checkDigit[MMD_CHECK_DIGIT_LEN],
   char primaryStreet[MMD_PRIMARYSTREET_LEN],
   char tabbedAddress[MMD_STREET_LEN],
   char lacs[MMD_LACS_LEN],
   char pm[4][MMD_PMB_LEN],
   char pm[4][MMD_PMB_RANGE_LEN]
   short *addressType
   double *longitude,
   double *latitude,
   short *precision
   char resultcode[MMD_RESULT_CODE],
   char censusBlockId[MMD_CENSUS_BLOCK_LEN]
   char *dpvConfCode
   char *dpvCmraIndicator
   char *dpvFalsePos
   char dpvFn1 [MM_DPVFOOTNOTE_LEN]
   char dpvFn2 [MM_DPVFOOTNOTE_LEN]
   char dpvFn3 [MM_DPVFOOTNOTE_LEN]
   char dpvFn4 [MM_DPVFOOTNOTE_LEN]
   char dpvFn5 [MM_DPVFOOTNOTE_LEN]
   char dpvFn6 [MM_DPVFOOTNOTE_LEN]
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] A context handle needed to reference candidates and match status from a geocode address call.</td>
</tr>
<tr>
<td>candidateNumber</td>
<td>[Input] The index of the desired candidate. The first candidate will have an index of 0.</td>
</tr>
<tr>
<td>firm</td>
<td>[Output] Candidate firm</td>
</tr>
<tr>
<td>streetAddress</td>
<td>[Output] Candidate street address</td>
</tr>
<tr>
<td>city</td>
<td>[Output] Candidate city</td>
</tr>
<tr>
<td>state</td>
<td>[Output] Candidate state</td>
</tr>
<tr>
<td>zip</td>
<td>[Output] Candidate ZIP code</td>
</tr>
<tr>
<td>recordType</td>
<td>[Output] Record type for candidate.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deliveryPoint</td>
<td>[Output] Delivery Point for candidate address.</td>
</tr>
<tr>
<td>carrierRoute</td>
<td>[Output] Carrier Route for candidate address.</td>
</tr>
<tr>
<td>checkDigit</td>
<td>[Output] Check Digit for candidate address.</td>
</tr>
<tr>
<td>primaryStreet</td>
<td>[Output] Primary street for candidate address.</td>
</tr>
<tr>
<td>tabbedAddress</td>
<td>[Output] Candidate tabbed address.</td>
</tr>
<tr>
<td>lacs</td>
<td>[Output] Candidate Lacs.</td>
</tr>
<tr>
<td>pmb</td>
<td>[Output] Candidate PMB.</td>
</tr>
<tr>
<td>pmb_range</td>
<td>[Output] Candidate PMB range/number.</td>
</tr>
<tr>
<td>addressType</td>
<td>[Output] Candidate MapMarker address type.</td>
</tr>
<tr>
<td>longitude</td>
<td>[Output] Longitude of geocoded point.</td>
</tr>
<tr>
<td>latitude</td>
<td>[Output] Latitude of geocoded point.</td>
</tr>
<tr>
<td>precision</td>
<td>[Output] Code representing the precision of the geocoded point.</td>
</tr>
<tr>
<td>resultcode</td>
<td>[Output] The candidate result code.</td>
</tr>
<tr>
<td>censusBlockId</td>
<td>[Output] The census id of the candidate.</td>
</tr>
</tbody>
</table>
| dpvConfCode       | [Output] The DPV return code for the candidate. The return code can be one of the following:  
|                   | N–This address does not exist.                                               |
|                   | Y–This address exists.                                                       |
|                   | S–The street address exists, but the unit does not.                         |
|                   | D–Address is incomplete (highrise with no unit, or rural route with no box number. |
| dpvCmraIndicator  | [Output] Indicates if the candidate belongs to a Commercial Mail Receiving Agency (CMRA). |
| dpvFalsePos       | [Output] If true, DPV has detected a condition in which the candidate appears to be artificially generated and not a legitimately obtained address. DPV will shut down immediately in this instance. |
| dpvfn1            | [Output] Field that may be populated with a standard USPS footnote code. Each of the codes is described below. |
| dpvfn2            | [Output] Field that may be populated with a standard USPS footnote code. Each of the codes is described below. |
| dpvfn3            | [Output] Field that may be populated with a standard USPS footnote code. Each of the codes is described below. |
| dpvfn4            | [Output] Field that may be populated with a standard USPS footnote code. Each of the codes is described below. |
Footnote Codes

AA—Input address matched to the ZIP+4 file.
A1—Input address not matched to the ZIP+4 file.
BB—Input address matched to DPV (all components).
CC—Input address primary number matched to DPV but secondary number not matched (present but invalid).
N1—Input address primary number matched to DPV but highrise address missing secondary number.
M1—Input address primary number missing.
M3—Input address primary number invalid.
P1—Input address missing PO, RR, or HC box number.
RR—Input address matched to CMRA.
R1—Input address matched to CMRA but secondary number not present.

Returns
The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

GeocodeGetErrorText()

Purpose
Given a valid MapMarker error code this method returns a string giving a short text description of the error. See Appendix H: MapMarker GeoEngine Error Codes, for a full list of error codes.

Syntax
```
long GeocodeGetErrorText(
    short errorCode,
    char errorString[MMD_ERROR_STRING_LEN])
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns
The call returns 0 if successful, and 1 if failed.
**GeocodeGetHwyExitCandidate()**

**Purpose**

Given a zero-based index number, this method returns the highway name, exit number, and/or exit name, as well as the ZIP Code, city, and coordinates of the exit location. This method cannot be used until its corresponding call, GeocodeHwyExit(), has been invoked. Using the count from that function, the individual candidates can be retrieved from GeocodeGetHwyExitCandidate(). The first candidate is 0.

**Syntax**

```c
long GeocodeGetHwyExitCandidate(
    short index,
    unsigned char hwyTypePre[MMD_HWY_PRETYPE_LEN],
    unsigned char hwyName[MMD_HWY_NAME_LEN],
    unsigned char hwyTypeSuf[MMD_HWY_POSTTYPE_LEN],
    unsigned char hwyDir[MMD_HWYDIR_LEN],
    unsigned char exitNum[MMD_EXIT_NUM_LEN],
    unsigned char exitName[MMD_HWY_EXIT_NAME_LEN],
    unsigned char city[MMD_CITY_LEN],
    unsigned char state[MMD_STATE_LEN],
    unsigned char zipcode[MMD_ZIP_LEN],
    double *longitude,
    double *latitude)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>[Input] Index that indicates the candidate in the array that should be returned.</td>
</tr>
<tr>
<td>hwyTypePre</td>
<td>[Output] Type prefix of the matched highway.</td>
</tr>
<tr>
<td>hwyName</td>
<td>[Output] The name of the matched highway.</td>
</tr>
<tr>
<td>hwyTypeSuf</td>
<td>[Output] Type suffix of the matched highway.</td>
</tr>
<tr>
<td>hwyDir</td>
<td>[Output] The direction of the matched highway.</td>
</tr>
<tr>
<td>exitNum</td>
<td>[Output] The number of the matched exit.</td>
</tr>
<tr>
<td>exitName</td>
<td>[Output] The name of the matched exit.</td>
</tr>
<tr>
<td>city</td>
<td>[Output] The returned city name.</td>
</tr>
<tr>
<td>state</td>
<td>[Output] The returned state name.</td>
</tr>
<tr>
<td>longitude</td>
<td>[Output] The X coordinate of the returned point.</td>
</tr>
<tr>
<td>latitude</td>
<td>[Output] The Y coordinate of the returned point.</td>
</tr>
</tbody>
</table>
Returns
This call returns 0 if successful. An error is returned if errors in the server or engine are encountered. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

GeocodeGetServerVersion()

Purpose
This function returns the server version number.

Syntax
long GeocodeGetServerVersion(double * pVersion);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns
The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

GeocodeGetStatesFound()

Purpose
This call returns a space-separated list of states found in the Address Dictionary Path. The list is not necessarily of licensed states.

Syntax
long GeocodeGetStatesFound(char stateAbbList[160]);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateAbbList</td>
<td>[Output] A space-separated list of states found in the Address Dictionary path.</td>
</tr>
</tbody>
</table>

Returns
The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

GeocodeGetStatesLicensed()

Purpose
This call returns a space-separated list of states currently licensed.

Syntax
long GeocodeGetStatesLicensed(char stateAbbList[160]);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

**GeocodeHwyExit()**

Purpose

This method uses input criteria of highway name, exit information, and state, and returns a count of the number of Hwy records that matched the input criteria. A subsequent call to GeocodeGetHwyExitCandidate() retrieves the matched candidates.

Syntax

```c
long GeocodeHwyExit(
  char *hwyExit,
  char *state,
  short *count)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hwyExit</td>
<td>[Input] The highway and exit information in the form:</td>
</tr>
<tr>
<td></td>
<td>&lt;highway name&gt; &lt;highway directional&gt; EXIT&lt;exit number&gt;&lt;exit number suffix&gt;TO +&lt;exit name&gt;. Where:</td>
</tr>
<tr>
<td></td>
<td>highway name is the name of the highway name including type prefix or type</td>
</tr>
<tr>
<td></td>
<td>suffix (e.g., Highway 8, I-90, or Pennsylvania Turnpike.</td>
</tr>
<tr>
<td></td>
<td>highway directional is the direction of the highway, full or abbreviated</td>
</tr>
<tr>
<td></td>
<td>(e.g., E or East). Optional.</td>
</tr>
<tr>
<td></td>
<td>EXIT keyword is a case-insensitive keyword used to separate the highway</td>
</tr>
<tr>
<td></td>
<td>information from the exit information. Required.</td>
</tr>
<tr>
<td></td>
<td>exit number is the actual number of the exit. This is optional if there is</td>
</tr>
<tr>
<td></td>
<td>an exit name in the input.</td>
</tr>
<tr>
<td></td>
<td>exit number suffix is any non-numerical exit suffix (e.g., Exit 2E).</td>
</tr>
<tr>
<td></td>
<td>Optional, not used by MapMarker.</td>
</tr>
<tr>
<td></td>
<td>TO keyword is a case-insensitive keyword used to separate the exit name and</td>
</tr>
<tr>
<td></td>
<td>exit number. Optional.</td>
</tr>
<tr>
<td></td>
<td>exit name is the name of the exit (e.g., State Hwy 5). Optional, MapMarker</td>
</tr>
<tr>
<td></td>
<td>will match only to exact names when there is no exit number in the input or</td>
</tr>
<tr>
<td></td>
<td>there is no match to the exit number in the input.</td>
</tr>
<tr>
<td></td>
<td>Sample input:</td>
</tr>
<tr>
<td></td>
<td>I-87 EXIT 2E TO State Hwy 5</td>
</tr>
<tr>
<td></td>
<td>I-87 EXIT State Hwy 5</td>
</tr>
<tr>
<td></td>
<td>I-87 EXIT 2E</td>
</tr>
<tr>
<td></td>
<td>I-87 EXIT 2</td>
</tr>
<tr>
<td></td>
<td>I-87 EXIT 2E State Hwy</td>
</tr>
<tr>
<td></td>
<td>This input is used by the engine to look up the specified highway and exit</td>
</tr>
<tr>
<td></td>
<td>in the highway exit data table (hwyexits.dbf in the Address Dictionary.</td>
</tr>
<tr>
<td>state</td>
<td>[Input] Contains the state name, either in full or abbreviated.</td>
</tr>
<tr>
<td>count</td>
<td>[Output] The total number of candidates found.</td>
</tr>
</tbody>
</table>

Returns

This call returns 0 if successful, and -1 if the input format is not correct or missing required keywords. An error is returned if errors in the server or engine are encountered. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

GeocodeIsCandidateMultiUnit()

Purpose

This method determines whether a candidate address specified by a zero-based index contains multiple units.

Syntax

```c
long GeocodeIsCandidateMultiUnit(GEOCODE_HANDLE geocodeHandle,
                                   short index,
                                   short *IsMultiUnit)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Input] The handle returned from the Geocode call used.</td>
</tr>
<tr>
<td>index</td>
<td>[Input] The index value of the candidate to examine. The index is zero-based.</td>
</tr>
<tr>
<td>IsMultiUnit</td>
<td>[Output] Returns 1 if the street address contains multiple units; 0 if not.</td>
</tr>
</tbody>
</table>

Returns

The call returns 0 if successful; otherwise, a MapMarker error code is returned. See Appendix H: MapMarker GeoEngine Error Codes for a full listing of error codes.

GeocodePostalCentroid()

Purpose

This call can be used to get a ZIP Code or ZIP+4 centroid.

Syntax

```c
short GeocodePostalCentroid(char *zip, char *plus4, double *longitude, double *latitude, short *precision, char resultcode[MMD_RESULT_CODE])
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zip</td>
<td>[Input] Address component.</td>
</tr>
<tr>
<td>plus4</td>
<td>[Input] Address component.</td>
</tr>
<tr>
<td>longitude</td>
<td>[Output] Longitude of the zip centroid.</td>
</tr>
<tr>
<td>latitude</td>
<td>[Output] Latitude of the zip centroid.</td>
</tr>
<tr>
<td>precision</td>
<td>[Output] Precision of the coordinate. See geostd.h for a complete list of precision returns.</td>
</tr>
<tr>
<td>resultcode</td>
<td>[Output] Result code of the zip centroid.</td>
</tr>
</tbody>
</table>

Returns

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.
**GeocodePostalCentroidWithSerial()**

**Purpose**

This call can be used to get a ZIP Code or ZIP+4 centroid. Use this function instead of GeocodePostalCentroid() in applications communicating with a server that is initialized with a serial number.

**Syntax**

```c
long GeocodePostalCentroidWithSerial(char *zip,
char *plus4,
double *longitude,
double *latitude,
short *precision,
char resultcode[MMD_RESULT_CODE],
char *pSerialNumber);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zip</td>
<td>[Input] Address component.</td>
</tr>
<tr>
<td>plus4</td>
<td>[Input] Address component.</td>
</tr>
<tr>
<td>longitude</td>
<td>[Output] Longitude of the zip centroid.</td>
</tr>
<tr>
<td>latitude</td>
<td>[Output] Latitude of the zip centroid.</td>
</tr>
<tr>
<td>precision</td>
<td>[Output] Precision of the coordinate. See geostd.h for a complete list of</td>
</tr>
<tr>
<td></td>
<td>precision returns.</td>
</tr>
<tr>
<td>resultcode</td>
<td>[Output] Result code of the zip centroid.</td>
</tr>
<tr>
<td>pSerialNumber</td>
<td>[Input] The OEM serial number.</td>
</tr>
</tbody>
</table>

**Returns**

The call returns 0 if successful; otherwise, one of the MapMarker error codes is returned. See Appendix H: MapMarker GeoEngine Error Codes, for a full listing of error codes.

**GeocodeShutdown()**

**Purpose**

Shuts down the MapMarker Server.

**Note:** This terminates all active sessions. The normal procedure is to shut down the MapMarker Server using the Windows Service Control Manager.

**Syntax**

```c
void GeocodeShutdown()
```
This chapter provides information for developers who wish to create customized geocoding applications based on the MapMarker Geocoding Engine Application Programming Interface.

This API is for use with C, C++, Visual Basic, and MapBasic. Applications developed using ANSI C are portable between Windows and UNIX platforms. Included are the function calls, programming notes, and sample code to assist you. The MapMarker GeoEngine is packaged as a 32-bit Dynamic Link Library (mm32v10.dll).

In this chapter:

- **Creating a Geocoding Application** ....................................................... 266
- **Notes for MapBasic and Visual Basic Programmers** .......................... 277
- **Sample Code** .................................................................................. 278
- **MapMarker Geocoding Engine Function Calls** .................................. 278
Creating a Geocoding Application

The MapMarker Geocoding Engine API provides all the functionality needed to create a full-scale geocoding application. The API provides calls to do batch and interactive geocoding, as well as calls to do browsing and intersection matching. The easiest way to begin writing geocoding applications is to customize the included sample applications.

Note: The API does not support the creation of MapInfo tables or the creation of MapInfo point objects. However, this functionality is available to developers of MapBasic applications using standard MapBasic functions.

The MapMarker Geocoding Engine API supports the following:

- Geocoding in batch mode or interactively
- Matching to street-level, ZIP centroid, place name, and street intersections
- Matching to the MapMarker Address Dictionary and customized dictionaries
- Creating customized user dictionaries
- Converting to and from NAD83 and NAD27
- Returning candidate information found in the MapMarker Address Dictionary
- Setting matching restrictions and system preferences
- Geocoding with an evaluation copy of MapMarker

The following sections detail how to write an ANSI C geocoding application. The 32-bit Windows DLL was built using Microsoft's Visual C++ V 6.0. This DLL should be compatible with most Windows compilers.

Concepts

All of the functions in the API return a short value. They are associated with a predefined constant to make them easier to understand. For example, SUCCESS is predefined as 0. This short value can be used to determine whether the call was successful or not. If the call was successful, then the value SUCCESS is returned. If not, then one of the error codes in `geoerror.h` is returned. The caller should be sure to check the return value from a function every time. If one step of the geocoding process fails, then the remaining calls may or may not be valid, and could lead to confusing results.

The API uses the term postalCode and postalAddOnCode instead of ZIP Code and ZIP+4. This is for possible future internationalization.

Within the sample source code [...] appears. This signifies that some source code was left out because it did not pertain to the example. Sample code is included in \mapmarker\geoeng\samples\.
Initialization

Before any calls in the API can be used (GeoEngCheckDbAvailability() is the only exception) the Geocoding Engine needs to be initialized. Initialization involves finding the Address Dictionary files, as well as reading in the parsing and matching support files (geo_usa.*). The call GeoEngInit() is used to initialize the engine. It can be used to initialize any combination of Street, ZIP Code, or User Dictionary level geocoding. GeoEngInit() returns, as a parameter, a pointer to a GEO_ENG_HANDLE. This handle is used by every call in the API with the exception of GeoEngDbCheckAvailability(). It is how the Geocoding Engine keeps track of the current state of the application’s geocoding progress.

Once the application has completed geocoding, the GeoEngKill() is used to terminate geocoding. It simply closes all files and frees any resources. It is important to note that GeoEngKill() needs to be called before another call to GeoEngInit() is initiated.

The call GeoEngDbCheckAvailability() can be used before the call to GeoEngInit() to determine the availability of the various geocoding dictionaries. The call also tries to initialize the parsing and matching components of the engine so that if any of the support files are not available an error code is returned.

The following sample code obtains the Address Dictionary database path from the user’s registry, and then tries to initialize street and centroid geocoding. For other platforms the code simply asks the user to enter the database path at the command line prompt.

```c
/* For WINDOWS include these */
#if defined(_WINDOWS)||defined(_WIN32)
   #include <windows.h>
#endif
/* Include the files needed for the Geocoding Engine. */
#ifdef __WINDOWS
   #define GEODLL
#endif
#include <geo.h>
#include <geoerror.h>
/* Prototypes for local functions */
short InitializeEngine(pGEO_ENG_HANDLE);
[...] 
/*
* InitializeEngine - Starts up the MapMarker Geocoding Engine.
* For Windows it consults the user ini file. For other platforms
* it asks for input.
*/
short InitializeEngine(pGEO_ENG_HANDLE pGeoEngHandle)
{
    char dbPath[256] = "";
    char *dbName = "geo_usa";
    GEO_ENG_CFG_PARMS geoEngParms;
    short retCode;
    short dbFlags;
```
printf("Initializing the Geocoding Engine. This could take a while.\n");
printf("Please enter the full path of the MapMarker address dictionary\n");
gets (dbPath);
/*
* Check availability, we only initialize if STREET_DB and CENTROID_DB are available.
*/
retCode = GeoEngCheckDbAvailability(dbPath, dbName, NULL, &dbFlags);
if (retCode != SUCCESS)
{
    fprintf(stderr, "Error %d checking MapMarker dictionary availability\n",retCode);
    return FAILURE;
}
if (!(dbFlags & STREET_DB))
{
    fprintf(stderr, "Geocoder Database Street files not available\n");
    fprintf(stderr, "Please try again with the correct dbPath\n");
    return FAILURE;
}
if (!(dbFlags & CENTROID_DB))
{
    fprintf(stderr, "Geocoder Database Centroid files not available\n");
    fprintf(stderr, "Please try again with the correct dbPath\n");
    return FAILURE;
}
/*
* If I got here, then try and initialize the engine.
*/
retCode = GeoEngInit(dbPath, dbName, NULL,
                     STREET_DB | CENTROID_DB, pGeoEngHandle);
if (retCode != SUCCESS)
{
    fprintf(stderr, "Error %d initializing MapMarker\n",retCode);
    return FAILURE;
}
return SUCCESS;
} /* end InitializeEngine function */

A Note About Include Files

At the top of the sample application, the geo.h and geoerror.h files are included. These source files are supplied with the Geocoding Engine source code. The geo.h and geostd.h files are needed to allow your application to reference all the defined constants, structures, and prototypes for the API. The geoerror.h contains a complete list of error codes returned by the API and some specific constants for those codes.
Setting Runtime Configuration Parameters

The API enables the application to control the behavior of the geocoding as well as the placement of geocoded points in relation to the matched segment. This behavior is controlled through the use of the GEO_ENG_CFG_PARMS structure.

```
typedef struct
{
    double linearInset;
    double perpendicularSetback;
    DIST_UNIT distUnit;
    BOOLEAN relaxHouseNumber;
    BOOLEAN relaxPostalCode;
    BOOLEAN relaxName;
    BOOLEAN relaxFinance;
    BOOLEAN relaxFinanceInState;
    BOOLEAN matchIntersections;
    BOOLEAN useCassMode;
    short relaxDistance;
    BOOLEAN exactCityName;
    BOOLEAN preferUserDictionary;
} GEO_ENG_CFG_PARMS, *pGEO_ENG_CFG_PARMS;
```

Each element in the structure controls a specific behavior of the API as follows:

- If `relaxPostalCode` is TRUE, then the API searches the finance number of the given ZIP Code, as well as the finance number of the city and state. The matched address and the input address need not have the same ZIP Code. If the parameter is FALSE, then only the finance number of the ZIP Code is searched, and the match address and the input address need to have the same ZIP Code.

- If `relaxName` is TRUE, then the API does a SOUNDEX (sounds like) search for a given street name and chooses what it thinks is the best match. If the value is FALSE, then only an exact street name match is considered acceptable.

- The `relaxHouseNumber` member is used to tell the API that the matched address and the input address need not have the same house number. The API places the matched coordinate at the closest (sequentially) house number on the matched street. If no house numbers are available for the street, then the API geocodes the input address to the midpoint of the shape path.

- If `relaxFinance` is TRUE, then the API searches a wider area for the address. The size of this area is defined by `relaxDistance`. All finance areas that have centroids within `relaxDistance` are searched as well. If `relaxFinanceInState` is TRUE, then the widened search area (indicated by `relaxFinance` being TRUE) is limited to the current state. Finance areas in other states are not searched.

- If `matchIntersections` is TRUE, then intersection matching is enabled.

- If `exactCityName` is set to TRUE, then the API only considers an exact city name match as a close match.

- The `preferUserDictionary` setting, if set to TRUE, defers to the User Dictionary (UD) for all matches that are the same or similar to matches found in the Address Dictionary (AD).
If `useCassMode` is TRUE, then all of the other configuration parameters are assigned their default CASS values. The default CASS values are:

- `relaxHouseNumber = False`
- `relaxPostalCode = True`
- `relaxName = True`
- `relaxFinance = False`
- `relaxFinanceInState = False`
- `matchIntersections = False`
- `relaxDistance = 0`
- `exactCityName = False`
- `preferUserDictionary = False`

The `linearInset`, `perpendicularSetback`, and `distUnit` values are used to control the location of a point when geocoding requires interpolation along a street segment. The `linearInset` value can be used to set all points a given distance from an intersection. This can be used to prevent the first house on cross streets from sitting on top of one another. The `perpendicularSetback` value is used to move the interpolated point away from the center of the street. This keeps points on the left and right sides of the street segments. The `distUnit` value defines the distance unit used for `linearInset` and `perpendicularSetback`.

There are two calls supplied for manipulating the configuration parameters. They are `GeoEngSetCfgParms()` and `GeoEngGetCfgParms()`. It is recommended that the application get the current values before changing the values. This way the application only needs to modify the values in the structure that are going to change.

The following code sample obtains the current values, and then relaxes names and ZIP Codes, and requires house numbers.

```c
short InitializeEngine(pGEO_ENG_HANDLE pGeoEngHandle)
{
    char dbPath[256] = "";
    char *dbName = "geo_usa";
    GEO_ENG_CFG_PARMS geoEngParms;
    short retCode;
    short dbFlags;
    [...]
```
Matching the Address

The API function GeoEngMatchAddress() is the routine that attempts to geocode the input address. The input address needs to be supplied to the application and must contain the street name and the ZIP Code (although if ZIP Codes are relaxed, city and state suffice). The function returns SUCCESS if it was able to parse the address and able to try to match candidates against the input address. The matchStatus parameter can be used to determine the extent to which the address was matched. If the matchStatus is SINGLE_MATCH, then a single good-street match that met all the relaxation criteria was found. The application can then geocode the address using GeoEngGetCandidateCoords(). If matchStatus is MULTIPLE_MATCHES, then more than one single good street match was found. At this point the application can choose to geocode the address or use some of the clerical functions described later to allow interactive geocoding of the address. A matchStatus of NO_MATCHES means that some candidates were found but none were close matches that met the relaxation criteria. At this point the application could choose to not geocode the address or once again use the clerical functions to allow interactive geocoding. A NO_CANDIDATES matchStatus result means there was nothing in the Address Dictionary even close to the input address. Browsing would be the only option at this point.
short matchStatus;
GEO_ENG_HANDLE geoEngHandle;
ADDRESS inputAddress;
PARSED_ADDRESS cleanAddress;
short retCode;
retCode = GeoEngMatchAddress(geoEngHandle,&inputAddress,
 &cleanAddress,&matchStatus);

[...] else if (retCode == SUCCESS)
{
  switch (matchStatus)
  {
    case SINGLE_MATCH:
      [...] break;
    case MULTIPLE_MATCHES:
      [...] break;
    case NO_MATCHES:
      [...] break;
    case NO_CANDIDATES:
      [...] break;
    case SINGLE_INTERSECT_MATCH:
      [...] break;
    case MULTIPLE_INTERSECT_MATCH:
      [...] break;
    case NO_INTERSECT_MATCHES:
      [...] break;
    case NO_INTERSECT_CANDIDATES:
      [...] break;
    default:
      printf("Invalid match status = %d\n",matchStatus);
      break;
  }
}

Getting Coordinates

The API provides two functions for getting coordinates. The routine
GeoEngGetCandidateCoords() can be used after a call to GeoEngMatchAddress() that returns
SUCCESS and a matchStatus of SINGLE_MATCH, MULTIPLE_MATCHES, or NO_MATCH.
GeoEngGetCandidateCoords() needs to be called before any new calls to
GeoEngMatchAddress(). This is because the API maintains a list of candidates for the most recent
match attempt, and a call to GeoEngMatchAddress() replaces the current candidate list.
DOUBLE_POINT location;
short retCode;
short precision;
[...]
retCode = GeoEngGetCandidateCoords(geoEngHandle, 0,
    &precision, &location);
    
    if (retCode != SUCCESS)
    {
        fprintf (stderr, “Error %d getting candidate
            coordinates\n”, retCode);
        return retCode;
    }

The other API call for getting coordinates, GeoEngFindPostalCentroid(), can be used to obtain
a ZIP Code or a ZIP+4 centroid. The call must be passed a valid ZIP Code. The Plus4 is optional.
A NULL pointer or an empty string may be supplied for the ZIP+4.

DOUBLE_POINT location;
short retCode;
short precision;
retCode = GeoEngFindPostalCentroid(geoEngHandle, 
    pInputAddress->postalCode, 
    pInputAddress->postalAddOnCode, 
    &precision, &location, NULL);
    
    if (retCode != SUCCESS)
    {
        fprintf (stderr, “Error %d getting postal
            centroid\n”, retCode);
        return retCode;
    }
    
else
    {
        printf (“MapMarker Point = %f longitude, %f latitude\n”,
            location.x, location.y);
    }

Getting the Candidates

The API provides the caller of GeoEngMatchAddress() with the choice of viewing all the
candidates that were found. This can be helpful if building an interactive geocoder. It can also be
used to obtain candidate information for the given match such as Census Blocks, matched street
address, matched ZIP+4, matched USPS last line information (city, state), as well as the score of
how the various candidate fields matched. The caller should first obtain the number of candidates
in the current list. The call GeoEngGetCandidateCount() is used here. The call returns the
number of close matches, as well as the total number of candidates. A close match is a candidate
that was close on all the match fields, and met the relaxation criteria (see “Setting Runtime
Configuration Parameters” on page 269). Once the total number of candidates is known, then the
call GeoEngGetIndexedCandidate() can be used to obtain the candidate address.
unsigned short i;
unsigned short closeMatchCount;
unsigned short candidateCount;
CANDIDATE_ADDRESS matchAddress;
short retCode;
short precision;
DOUBLE_POINT location;

[...]
retCode = GeoEngGetCandidateCount(geoEngHandle,
                     &closeMatchCount, &candidateCount);
if (retCode != SUCCESS)
    {
        fprintf (stderr, "Error %d getting candidate
count\n",retCode);
        return retCode;
    }
[...]
else
    {
        for (i = 0; i < closeMatchCount; i++)
            {
                retCode = GeoEngGetIndexedCandidate(geoEngHandle, i,
                                                  &matchAddress);
                if (retCode != SUCCESS)
                    {
                        fprintf (stderr, "Error %d getting match
candidate\n",retCode);
                        return retCode;
                    }
            }
    }

It is important to note that if GeoEngMatchAddress() returns SUCCESS and a matchStatus of
SINGLE_MATCH or MULTIPLE_MATCHES, then the GeoEngGetIndexedCandidate() can
always be used with an index 0 to obtain the best match.

The returned candidates all have a house number chosen to be the closest available house
number on the street. If this is not good enough for the given application, or if the caller wishes to
allow the user to choose the correct house number range, then the call
GeoEngGetNextHouseRange() can be used to examine all the house number ranges available. The
house number ranges are a combination of TIGER street ranges, and USPS ZIP+4 ranges, so there
could be a large number of them for a given candidate. The call
GeoEngGetNextHouseRange() can be called for each candidate and returns
GEO_ENG_END_OF_DATA when done. The BOOLEAN parameter firstRange tells the API
whether the caller wants the first range for this candidate, or the next.
The following while loop demonstrates the use of this call.

```c
unsigned short i;
unsigned short candidateCount;
short retCode;
BOOLEAN bFirst;
HOUSE_RANGE houseRange;

for (i = 0; i < candidateCount; i++)
{
    /*
    * Now dump all the house ranges for the candidate
    */
    printf("\tHouse Ranges\n");
bFirst = TRUE;
while ((retCode = GeoEngGetNextHouseRange(geoEngHandle, i,
bFirst, &houseRange)) == SUCCESS)
{
    bFirst = FALSE;
    PrintHouseRange(&houseRange);
}
if (retCode != GEO_ENG_END_OF_DATA)
{
    fprintf(stderr, "Error %d from
GeoEngGetNextHouseRange\n", retCode);
    return retCode;
}

/*
* PrintHouseRange - prints out the house range in a nice format.
*/
void PrintHouseRange(pHOUSE_RANGE pHouseRange)
{
    printf("\tFrom %10.10s To %10.10s ZIP Code %s-%s\n",
        pHouseRange->fromHouse, pHouseRange->toHouse,
PHouseRange->postalCode, pHouseRange->postalAddOnCode);
}
```

If the user decides that they wish to geocode the address using a different house range, then the call `GeoEngGetIndexedCoords()` can be used. It takes the candidate index as well as the index of the house range, and returns the coordinate for the input house (closest house number on the range if available).

### Browsing the Address Dictionary

The API allows an application to browse through the actual streets and places in the Address Dictionary. Since the dictionary is organized by finance number, the caller can only view streets and places in a given finance number at a time. The supplied ZIP Code is used to determine the finance number to browse. The calls to browse the dictionary are `GenEngBrowseStreet()`, and `GeoEngBrowsePlace()`. The caller needs to provide a street filter and a ZIP Code before calling these routines. This is done using the call `GeoEngBrowseSetFilter()`. Any street or place that
begins with the filter string, and is in the correct finance number for the given ZIP Code, is returned. Thus if the filter was “MA” then all streets beginning with the letters “MA” in the finance number of the ZIP Code would be returned.

```c
BOOLEAN bFirst;
short retCode;
char browseFilter[256];
char ZIPCode[256];
BROWSED_STREET browsedStreet;

[...] 
bFirst = FALSE;
retCode = GeoEngBrowseSetFilter(geoEngHandle, ZIPCode, browseFilter);
if (retCode != SUCCESS)
{
    fprintf(stderr, "Error %d from GeoEngBrowseSetFilter\n", retCode);
    return retCode;
}
while((retCode = GeoEngBrowseStreet(geoEngHandle, &browsedStreet)) == SUCCESS)
{
    printf("\tStreet %s %s %s %s %s\n", browsedStreet.preDir, browsedStreet.preType, browsedStreet.name, browsedStreet.postType, browsedStreet.postDir);
}
if (retCode != GEO_ENG_END_OF_DATA)
{
    fprintf(stderr, "Error %d from GeoEngBrowseStreet\n");
    return retCode;
}
[...]```

Include Files

The MapMarker Geocoding Engine API comes with include files for ANSI C applications. The files can be found in the \mapmarker\geoeng\include subdirectory created when you install MapMarker.

For ANSI C, the application needs to include the files geo.h and geoerror.h. The geo.h include file contains all the function prototypes, defined constants, and structure declarations needed when using the API. The geoerror.h contains a complete list of error codes returned by the API and some specific constants for those codes.

Compiling and Linking a Windows Application

Use the following command line flags to compile a Window’s ANSI C application:

- `/AL` – Select the large memory model.
- `/D GEODLL` – Define the constant GEODLL so prototype has the _export modifier.
The application then needs to link against the 32-bit MapMarker library \mm32v10.lib and is placed in the \mapmarker\geoeng\lib subdirectory that is created when MapMarker is installed. The application needs to increase the stack size to at least 20K. This can be done using the STACKSIZE argument in the Windows DEF file.

STACKSIZE 20480

Notes for MapBasic and Visual Basic Programmers

In addition to the C interfaces, we have provided interfaces to the MapMarker API for MapBasic and Visual Basic in the directory \mapmarker\geoeng\include. MapBasic programmers should include MIGEO32.DEF in their programs, and Visual Basic programmers should use MIGEO.BAS. We have also provided MapBasic and Visual Basic example programs that demonstrate how to use the MapMarker API with these development environments. The examples are found in \mapmarker\geoeng\samples\. Programmers using other similar development environments should use these interface files and examples as a guide for developing interfaces in their environments.

There are a few special considerations when using the MapMarker API from MapBasic, Visual Basic, or similar environments. Make sure that parameters passed to the MapMarker Engine (DLL) are declared and passed correctly. In MapBasic and Visual Basic, strings included in structures passed to DLLs must be declared with fixed length. However, strings passed as direct parameters to MapMarker routines should not be declared as fixed length. This means that a string passed to one routine as part of a structure and to another as a direct parameter must be assigned to an intermediary local string variable of the correct type before being passed to the second routine.

MapBasic and Visual Basic programmers must also take care to always pass valid MapMarker engine pointers to the MapMarker engine. When you call GeoEngInit(), the returned value is a pointer to the memory used by that instance of the MapMarker engine. Though neither MapBasic nor Visual Basic have the concept of pointer variables, you can use a long integer (an integer in MapBasic) to store the pointer so that it can be passed to the other routines that require it. The problem is that passing a variable that does not reference a valid instance of the MapMarker engine causes an error or even a system crash.

Things to watch out for include:

- Do not call any MapMarker routines after calling GeoEngKill().
- Be sure to pass the correct variable to the DLL.
- In Visual Basic, be sure to use the Option Explicit directive to force VB to prevent the use of undeclared variables. Without this setting, VB allows you to declare new variables simply by using them. In this situation, if you incorrectly type the name of the GeoEngine pointer in the parameter list of a call to a MapMarker routine, VB creates a new variable by that name with a value of zero and passes it to the DLL. This usually results in a system crash.

Note: Option Explicit is declared in the file MIGEO.BAS provided with MapMarker.
Sample Code

Examples of programming code in several languages are included in MapMarker. These samples illustrate the function of the MapMarker GeoEngine. These are located in the \
mapmarker\geoeng\samples directory.

The sample under \MFC is a C++ example created using Developer's Studio 6.0.

Note: The files geo.h and geoerror.h are C header files.

MapMarker Geocoding Engine Function Calls

The following pages contain detailed syntax, parameter and return descriptions, as well as structures and defines. Use the list below to reference specific calls.

- GeoEngBrowseAirportByState() ........................................... 280
  Retrieves airport information for a given state.
- GeoEngBrowsePlace() .................................................. 282
  Retrieves a place name from the Address Dictionary.
- GeoEngBrowseSetFilter() .................................................. 283
  Sets the filter for browsing within the Address Dictionary.
- GeoEngBrowseStreet() ................................................... 284
  Retrieves the next street address from the Address Dictionary.
- GeoEngBuildResultCode() .............................................. 286
  Retrieves a georesult code.
- GeoEngCheckDbAvailability() .......................................... 287
  Checks database availability.
- GeoEngCheckDbAvailabilityWithLicense() ................................. 289
  Checks database availability for restricted geocoding.
- GeoEngFindPostalCentroid() ........................................... 291
  Returns postal centroid coordinates.
- GeoEngGetAirport() ...................................................... 294
  Returns airport location information.
- GeoEngGetCandidateCoords() .......................................... 297
  Geocodes the user address that was previously supplied in the most recent call to GeoEngMatchAddress().
- GeoEngGetCandidateCount() ............................................ 299
  Returns the number of match candidates.
- GeoEngGetCandidateDpvInfo() ......................................... 300
  Returns the DPV information for a candidate.
- GeoEngGetCfgParms() ................................................... 300
  Returns current run time parameters for the geocoding engine.
- GeoEngGetDpvMode() ..................................................... 302
  Returns the current status of DPV Mode: on or off.
- GeoEngGetErrorText() .................................................. 302
  Returns a description of a given error.
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoEngGetIndexedCandidate()</td>
<td>Returns a specified match candidate.</td>
</tr>
<tr>
<td>GeoEngGetIndexedCoords()</td>
<td>Geocodes a user address.</td>
</tr>
<tr>
<td>GeoEngGetIndexedHwyExit()</td>
<td>Returns highway exit information of the matched candidate using a zero-based index.</td>
</tr>
<tr>
<td>GeoEngGetNextHouseRange()</td>
<td>Returns the next street house range for a match candidate.</td>
</tr>
<tr>
<td>GeoEngGetStatesFound()</td>
<td>Returns states in Address Dictionary Path.</td>
</tr>
<tr>
<td>GeoEngGetStatesLicensed()</td>
<td>Returns licensed states.</td>
</tr>
<tr>
<td>GeoEngGetVersion()</td>
<td>Returns the engine version number.</td>
</tr>
<tr>
<td>GeoEngGetZIPCountyFIPSList()</td>
<td>Returns FIPS codes and counties for an input ZIP Code.</td>
</tr>
<tr>
<td>GeoEngInit()</td>
<td>Initializes the Geocoding Engine.</td>
</tr>
<tr>
<td>GeoEngInitWithLicense()</td>
<td>Initializes the Geocoding Engine with License.</td>
</tr>
<tr>
<td>GeoEngIsCandidateMultiUnit()</td>
<td>Determines if a candidate address in the candidate array contains multiple units.</td>
</tr>
<tr>
<td>GeoEngIsEval()</td>
<td>Checks if the Address Dictionary is an evaluation copy.</td>
</tr>
<tr>
<td>GeoEngKill()</td>
<td>Shuts down the Geocoding Engine.</td>
</tr>
<tr>
<td>GeoEngMatchAddress()</td>
<td>Matches a user address against the Address Dictionary.</td>
</tr>
<tr>
<td>GeoEngMatchFormattedAddress()</td>
<td>Matches a broken-down user address against the Address Dictionary.</td>
</tr>
<tr>
<td>GeoEngMatchHwyExit()</td>
<td>Returns the number of matched candidates, if any, using highway exit &amp; state criteria.</td>
</tr>
<tr>
<td>GeoEngNAD27ToNAD83()</td>
<td>Transforms coordinates from NAD27 to NAD83 Datum.</td>
</tr>
<tr>
<td>GeoEngNAD83ToNAD27()</td>
<td>Transforms coordinates from NAD83 to NAD27 Datum.</td>
</tr>
<tr>
<td>GeoEngParseLastLine()</td>
<td>Parses a last line into city, state, and ZIP Code components.</td>
</tr>
<tr>
<td>GeoEngSetCfgParms()</td>
<td>Sets run time parameters for the geocoding engine.</td>
</tr>
<tr>
<td>GeoEngSetDpvMode()</td>
<td>Sets the GeoEngine to geocode in DPV mode.</td>
</tr>
<tr>
<td>GeoEngSetInteractive()</td>
<td>Sets the GeoEngine to do interactive matching.</td>
</tr>
</tbody>
</table>
GeoEngBrowseAirportByState()

Purpose
This call retrieves airport information for a given state. GeoEngBrowseAirportByState() retrieves a record containing a point location and other information for all airports listed for the given state when passed a valid State abbreviation. Use successive calls to retrieve relevant records. The function returns GEO_ENG_END_OF_DATA when there are no more records for that state or the end of data is encountered.

Syntax
short GeoEngBrowseAirportByState(GEO_ENG_HANDLE geoEngHandle, 
char * pState, 
pMEM_AIRPORT_REC pAirport_rec)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pAirport_rec</td>
<td>[Output] Pointer to structure to receive airport information.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_END_OF_DATA</td>
<td>11</td>
<td>No more data available from the requested set.</td>
</tr>
<tr>
<td>GEO_ENG_NO_APS</td>
<td>30</td>
<td>No airports found in given state.</td>
</tr>
<tr>
<td>GEO_ENG_LOW_LEVEL_AP_ERROR</td>
<td>31</td>
<td>Low level operation error.</td>
</tr>
</tbody>
</table>
Structures and Defines

#define LOC_ID_KEY_LEN    4
#define LOC_ID_LEN        4
#define AIRPORT_NAME_LEN  55
#define CFCC_LEN          3
#define USE_LEN           2
#define OWNERNAME_LEN     29
#define CONGES_LEV_LEN    1
#define SEVR_LEV_LEN      2
#define HUB_SIZE_LEN      1
#define TOWER_TYPE_LEN    1
#define STATE_LEN         2
#define STATE_KEY_LEN     2
#define FIPS_CNTY_LEN     5

typedef struct
{
    char locId[LOC_ID_LEN + 1];
    char airportName[AIRPORT_NAME_LEN + 1];
    char CFCC[CFCC_LEN + 1];
    char use[USE_LEN + 1];
    char ownerName[OWNERNAME_LEN + 1];
    char congesLev[CONGES_LEV_LEN + 1];
    char servLev[SEVR_LEV_LEN + 1];
    char hubSize[HUB_SIZE_LEN + 1];
    char towerType[TOWER_TYPE_LEN + 1];
    char state[STATE_LEN + 1];
    char FIPSCnty[FIPS_CNTY_LEN + 1];
    double X;
    double Y;
    int id;
    int elevation;
    int lrgCertEnp;
    int commEnp;
    int airTaxiEnp;
    int foreignEnp;
    int inTranEnp;
    int iRecNo;
} MEM_AIRPORT_REC, *pMEM_AIRPORT_REC;
GeoEngBrowsePlace()

Purpose

This call retrieves the next place name from the Address Dictionary which should previously have been positioned via GeoEngBrowseSetFilter() without matching a user address. The current database position is updated so that successive calls to this function step through the place names that satisfy the filter string in the postal region that was specified in the most recent call to GeoEngBrowseSetFilter(). Repeated use of this function eventually returns GEO_ENG_END_OF_DATA when there are no more place names that satisfy the filter. Note that this function does not support browsing with user dictionaries.

Syntax

    short GeoEngBrowsePlace(GEO_ENG_HANDLE geoEngHandle,
                        pBROWSED_PLACE pPlaceName);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pPlaceName</td>
<td>[Output] A pointer to a variable that receives the browsed place name.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_END_OF_DATA</td>
<td>11</td>
<td>No more data available from the requested set.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENG_NO_DATA_AVAILABLE</td>
<td>15</td>
<td>Internal database inconsistency.</td>
</tr>
</tbody>
</table>
Structures and Defines

/*
 * Defined constants for character array sizes in a BROWSED_PLACE
 * structure
 */
#define MAX_PLACE_NAME_LEN 35
typedef struct
{
    char name[MAX_PLACE_NAME_LEN];
} BROWSED_PLACE, *pBROWSED_PLACE;

GeoEngBrowseSetFilter()

Purpose

This call sets the filter for browsing within the Address Dictionary by means of the supplied filter string and postal region. This call must immediately precede the first use of GeoEngBrowseStreet() or GeoEngBrowsePlace(). The next call to a function other than these destroys the filter information. If this call is not issued, access to the database is random. Calls to GeoEngBrowseStreet() cannot be interspersed with calls to GeoEngBrowsePlace(). If browsing of both is desired, you should complete browsing all streets or all places and then call GeoEngBrowseSetFilter() again before continuing with the other type of browsing.

Syntax

    short GeoEngBrowseSetFilter(GEO_ENG_HANDLE geoEngHandle,
        char *pPostalCode,
        char *pFilterString);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pPostalCode</td>
<td>[Input] A string containing the postal code.</td>
</tr>
<tr>
<td>pFilterString</td>
<td>[Input] A string whose contents specify the leading characters in all street or place names that can be retrieved using GeoEngBrowseStreet() or GeoEngBrowsePlace().</td>
</tr>
</tbody>
</table>

Note: Once the filter has been used by calling GeoEngBrowseStreet() or GeoEngBrowsePlace() then it needs to be called again for any other browse calls.
Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading from database file.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>9</td>
<td>Error accessing database index.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_DATABASE_ERR</td>
<td>10</td>
<td>Invalid database contents.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_POSTAL_CODE</td>
<td>12</td>
<td>Invalid postal code.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_INPUT_ADDRESS</td>
<td>14</td>
<td>Input address is invalid or missing fields.</td>
</tr>
</tbody>
</table>

GeoEngBrowseStreet()

Purpose
This call retrieves the next street address from the Address Dictionary. The database cursor should previously have been positioned via a call to GeoEngBrowseSetFilter(). The current database position is updated so that successive calls to this function step through the streets that satisfy the filter string in the postal region that was specified in the most recent call to GeoEngBrowseSetFilter(). Repeated use of this function eventually returns GEO_ENG_END_OF_DATA when there are no more streets that satisfy the filter. Note that this function does not support browsing with user dictionaries.

Syntax

```c
short GeoEngBrowseStreet(GEO_ENG_HANDLE geoEngHandle,
                        pBROWSED_STREET pStreetInfo);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pStreetInfo</td>
<td>[Output] A pointer to a variable that receives the browsed street information from the database.</td>
</tr>
</tbody>
</table>
Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>9</td>
<td>Error accessing database index.</td>
</tr>
<tr>
<td>GEO_ENG_END_OF_DATA</td>
<td>11</td>
<td>No more data available from the requested set.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENG_NO_DATA_AVAILABLE</td>
<td>15</td>
<td>Internal database inconsistency.</td>
</tr>
<tr>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>1075</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
<tr>
<td>GEO_ENG_UNDEFINED_SEARCH</td>
<td>1077</td>
<td>GeoEngBrowseSetFilter was not called.</td>
</tr>
</tbody>
</table>

Structures and Defines

```c
/*
 * Defined constants for the character array sizes in all the
 * BROWSED_STREET structure.
 */
#define MAX_STREET_PREDIR_LEN 3
#define MAX_STREET_PRETYPE_LEN 11
#define MAX_STREET_NAME_LEN 29
#define MAX_STREET_POSTTYPE_LEN 5
#define MAX_STREET_POSTDIR_LEN 3

typedef struct
{
    char preDir[MAX_STREET_PREDIR_LEN];
    char preType[MAX_STREET_PRETYPE_LEN];
    char name[MAX_STREET_NAME_LEN];
    char postType[MAX_STREET_POSTTYPE_LEN];
    char postDir[MAX_STREET_POSTDIR_LEN];
} BROWSED_STREET, *pBROWSED_STREET;
```
GeoEngBuildResultCode()

Purpose
This call retrieves a result code. The input address, candidate precision, candidate match status, and street or ZIP Code level flag needs to be supplied to the application. It returns SUCCESS if it was able to produce a result code. The `postalGeocodeFlag` parameter should be set to FALSE to return a street-level result code or set to TRUE to return a ZIP Code level result code. If returning a ZIP Code level result code the candidate `matchStatus` parameter should be set to zero.

Syntax

```c
short GeoEngBuildResultCode(pCANDIDATE_ADDRESS pCandidateAddress,
    pADDRESS pAddr,
    short matchprecision,
    short matchStatus,
    BOOLEAN postalGeocodeFlag,
    char * georesult);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pCandidateAddress</td>
<td>[Input] The address of the current candidate address structure. This value can be retrieved from <code>GeoEngGetIndexedCandidate()</code></td>
</tr>
<tr>
<td>pAddr</td>
<td>[Input] The address of the input Address structure. This value is the same as that used for input to <code>GeoEngMatchAddress()</code></td>
</tr>
<tr>
<td>matchprecision</td>
<td>[Input] The coordinate precision level of the match. This is retrieved from <code>GeoEngFindPostalCentroid()</code> for ZIP Code matching or <code>GeoEngGetCandidateCoords()</code> for street-level matching.</td>
</tr>
<tr>
<td>matchStatus</td>
<td>[Input] The type of street match for the candidate (e.g., SINGLE_MATCH, MULTIPLE_MATCHES). This value can be retrieved from <code>GeoEngMatchAddress()</code> or <code>GeoEngMatchFormattedAddress()</code>. This parameter is ignored when doing ZIP Code level geocoding, therefore any integer value can be used.</td>
</tr>
<tr>
<td>postalGeocodeFlag</td>
<td>[Input] This indicates whether Street or ZIP Code level geocoding was done. TRUE indicates ZIP Code level geocoding; FALSE indicates street-level geocoding.</td>
</tr>
<tr>
<td>georesult</td>
<td>[Output] A pointer to variable returning the result code. Note: This function does not generate a &quot;NG&quot; or &quot;ND&quot; result code.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function.</td>
</tr>
</tbody>
</table>
GeoEngCheckDbAvailability()

Purpose
This call checks database availability. It checks which geocoding database components are
available on a given search path using a specific dictionary name.

Syntax
short GeoEngCheckDbAvailability(char *pSearchPath,
char *pDbName,
char *pUserDictionaryPath,
short *pDbFlags);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSearchPath</td>
<td>[Input] The directory search path to locate the dictionary files. This is a string containing directory paths separated by semicolons.</td>
</tr>
<tr>
<td>pDbName</td>
<td>[Input] The base name to use in constructing the names of the dictionary files.</td>
</tr>
<tr>
<td>pUserDictionaryPath</td>
<td>[Input] The full path to the user dictionary files.</td>
</tr>
</tbody>
</table>
| pDbFlags                | [Output] A pointer to an integer that receives a value indicating which dictionary components were found. The value is the bitwise OR of the following defined constants and indicates which dictionary components can be initialized using the specified search path and Address Dictionary name.  
  • 1 – STREET_DB
  • 2 – CENTROID_DB
  • 3 – STREET_DB and CENTROID_DB
  • 4 – USER_DB
  • 5 – STREET_DB and USER_DB
  • 6 – CENTROID_DB and USER_DB
  • 7 – STREET_DB, CENTROID_DB, and USER_DB |

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>Memory allocation of internal structure failed.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_NOT_FOUND_ERR</td>
<td>4</td>
<td>Input file was not found.</td>
</tr>
<tr>
<td>Return</td>
<td>Return #</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GEO_ENG_FILE_OPEN_ERR</td>
<td>5</td>
<td>Error opening sinames, cityfin, usafin, citylin, file.tbl, license file, etc.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading sinames, cityfin, usafin, citylin, file.tbl, license file, etc.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_OPEN_ERR</td>
<td>8</td>
<td>Error while trying to open a dictionary index.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>9</td>
<td>Error while trying to read a dictionary index.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_LICFILE_ERR</td>
<td>18</td>
<td>Invalid license file.</td>
</tr>
<tr>
<td>GEO_ENG_PARSE_INIT_ERR</td>
<td>307</td>
<td>Parsing file (geo_usa.*) missing or bad.</td>
</tr>
<tr>
<td>GEO_ENG_MISSING_DATABASE_ERR</td>
<td>1078</td>
<td>Data for street not licensed or not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_LICFILE</td>
<td>1079</td>
<td>License file not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_FILEID_TABLE</td>
<td>1080</td>
<td>File.tbl not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_ADR_ADX</td>
<td>1081</td>
<td>.adr/.adx files not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_BAD_HEADER</td>
<td>1082</td>
<td>Unable to read the header of file.tbl.</td>
</tr>
<tr>
<td>GEO_ENG_INVALID_SERIAL_NUMBER</td>
<td>1085</td>
<td>Invalid serial number.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_USER_DICT_INIT_ERR</td>
<td>1087</td>
<td>Error finding user dictionary files.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_FILE</td>
<td>1089</td>
<td>Citylin.idx not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_OPEN</td>
<td>1090</td>
<td>Could not open citylin.idx.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_LICFILE_EXPIRED</td>
<td>1091</td>
<td>Trial license has expired.</td>
</tr>
<tr>
<td>GEO_ENG_MAX_USER_DICT_EXCEEDED_ERR</td>
<td>1092</td>
<td>Exceeded limit on number of user dictionaries allowed.</td>
</tr>
<tr>
<td>GEO_ENG_SINAMES_INDEX_FILE</td>
<td>1093</td>
<td>Could not find sinames.idx.</td>
</tr>
<tr>
<td>GEO_ENG_SINAMES_INDEX_OPEN</td>
<td>1094</td>
<td>Could not open sinames.idx.</td>
</tr>
<tr>
<td>GEO_ENG_Z9_CEN_FILE_NOT_FOUND_ERR</td>
<td>1095</td>
<td>ZIP centroid (.cen) files not found.</td>
</tr>
<tr>
<td>GEO_ENG_UNIQUE_ZIP_INDEX_FILE</td>
<td>1097</td>
<td>Could not find cityunz.idx.</td>
</tr>
<tr>
<td>GEO_ENG_UNIQUE_ZIP_INDEX_OPEN</td>
<td>1098</td>
<td>Could not open cityunz.idx.</td>
</tr>
</tbody>
</table>
GeoEngCheckDbAvailabilityWithLicense()

Purpose

This call checks database availability for restricted geocoding.

GeoEngCheckDbAvailabilityWithLicense() checks which geocoding database components are available on a given search path using a specific database name, serial number, and license file extension.

Syntax

```c
short GeoEngCheckDbAvailabilityWithLicense(char *pSearchPath,
                                            char *pDbName,
                                            char *pUserDictPath,
                                            short *pDbFlags
                                            char *pSerialNumber
                                            char licenseExt[4]);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSearchPath</td>
<td>[Input] Directory search path to locate the dictionary files. This is a string containing directory paths separated by semicolons.</td>
</tr>
<tr>
<td>pDbName</td>
<td>[Input] Base name to use in constructing the names of the dictionary files.</td>
</tr>
<tr>
<td>pUserDictPath</td>
<td>[Input] The full path to the user dictionary files.</td>
</tr>
</tbody>
</table>

Return

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_ENG_URBANIZ_ZIP_INDEX_FILE</td>
<td>1101</td>
<td>Could not find cityurb.idx.</td>
</tr>
<tr>
<td>GEO_ENG_URBANIZ_ZIP_INDEX_OPEN</td>
<td>1102</td>
<td>Could not open cityurb.idx.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_NOT_FOUND_ERR</td>
<td>1228</td>
<td>Zipmastr.cdb or Zipmastr.jdx not found.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_READ_ERR</td>
<td>1230</td>
<td>Error reading Zipmastr.cdb or Zipmastr.jdx.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_OPEN_ERR</td>
<td>1232</td>
<td>Error opening Zipmastr.cdb or Zipmastr.jdx.</td>
</tr>
</tbody>
</table>
### Parameter Description

**pDbFlags**  
[Output] A pointer to an integer that receives a value indicating which dictionary components were found. The value is the bitwise OR of the following defined constants and indicates which dictionary components can be initialized using the specified search path and Address Dictionary name.
- 1 – STREET_DB
- 2 – CENTROID_DB
- 3 – STREET_DB and CENTROID_DB
- 4 – USER_DB
- 5 – STREET_DB and USER_DB
- 6 – CENTROID_DB and USER_DB
- 7 – STREET_DB, CENTROID_DB, and USER_DB

**pSerialNumber**  
[Input] Serial number to use.

**licenseExt**  
[Input] The license file extension.

### Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>Memory allocation of internal structure failed.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_NOT_FOUND_ERR</td>
<td>4</td>
<td>Input file was not found.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_OPEN_ERR</td>
<td>5</td>
<td>Error opening sinames, cityfin, usafin, cityline, file.tbl, license file, etc.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading sinames, cityfin, usafin, cityline, file.tbl, license file, etc.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_OPEN_ERR</td>
<td>8</td>
<td>Error while trying to open a dictionary index.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>9</td>
<td>Error while trying to read a dictionary index.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_LICFILE_ERR</td>
<td>18</td>
<td>Invalid license file.</td>
</tr>
<tr>
<td>GEO_ENG_PARSE_INIT_ERR</td>
<td>307</td>
<td>Parsing file (geo_usa.*) missing or bad.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_LICFILE</td>
<td>1079</td>
<td>License file not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_FILEID_TABLE</td>
<td>1080</td>
<td>File.tbl not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_ADR_ADX</td>
<td>1081</td>
<td>.adr/.adx files not found.</td>
</tr>
</tbody>
</table>
GeoEngFindPostalCentroid()

Purpose
This geocodes by retrieving postal centroid coordinates, based on the supplied postal code and postal add-on code.

Syntax
```c
short GeoEngFindPostalCentroid(GEO_ENG_HANDLE geoEngHandle,
    char *pPostalCode,
    char *pPostalAddOnCode,
    short *pCoordPrecision,
    pDOUBLE_POINT pLocation);
    pCANDIDATE_ADDRESS pCandidate);
```

Return

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_ENG_DBQ_BAD_HEADER</td>
<td>1082</td>
<td>Unable to read the header of file.tbl.</td>
</tr>
<tr>
<td>GEO_ENG_INVALID_SERIAL_NUMBER</td>
<td>1085</td>
<td>Invalid serial number.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_USER_DICT_INIT_ERR</td>
<td>1087</td>
<td>Error finding user dictionary files.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_FILE</td>
<td>1089</td>
<td>Cityline.idx not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_OPEN</td>
<td>1090</td>
<td>Could not open cityline.idx.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_LICFILE_EXPIRED</td>
<td>1091</td>
<td>Trial license has expired.</td>
</tr>
<tr>
<td>GEO_ENG_MAX_USER_DICT_EXCEEDED_ERR</td>
<td>1092</td>
<td>Exceeded limit on number of user dictionaries allowed.</td>
</tr>
<tr>
<td>GEO_ENG_SINAMES_INDEX_FILE</td>
<td>1093</td>
<td>Could not find sinames.idx.</td>
</tr>
<tr>
<td>GEO_ENG_SINAMES_INDEX_OPEN</td>
<td>1094</td>
<td>Could not open sinames.idx.</td>
</tr>
<tr>
<td>GEO_ENG_Z9_CEN_FILE_NOT_FOUND_ERR</td>
<td>1095</td>
<td>ZIP centroid (.cen) files not found.</td>
</tr>
<tr>
<td>GEO_ENG_UNIQUE ZIP_INDEX_FILE</td>
<td>1097</td>
<td>Could not find cityunz.idx.</td>
</tr>
<tr>
<td>GEO_ENG_UNIQUE ZIP_INDEX_OPEN</td>
<td>1098</td>
<td>Could not open cityunz.idx.</td>
</tr>
<tr>
<td>GEO_ENG_URBANIZ ZIP_INDEX_FILE</td>
<td>1101</td>
<td>Could not find cityurb.idx.</td>
</tr>
<tr>
<td>GEO_ENG_URBANIZ ZIP_INDEX_OPEN</td>
<td>1102</td>
<td>Could not open cityurb.idx.</td>
</tr>
<tr>
<td>GEO_ENG ZIP MASTER_FILE_NOT_FOUND_ERR</td>
<td>1228</td>
<td>Zipmastr.cdb or Zipmastr.jdx not found.</td>
</tr>
<tr>
<td>GEO_ENG ZIP MASTER_FILE_READ_ERR</td>
<td>1230</td>
<td>Error reading Zipmastr.cdb or Zipmastr.jdx.</td>
</tr>
<tr>
<td>GEO_ENG ZIP MASTER_FILE_OPEN_ERR</td>
<td>1232</td>
<td>Error opening Zipmastr.cdb or Zipmastr.jdx.</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pPostalCode</td>
<td>[Input] A string containing the postal code.</td>
</tr>
<tr>
<td>pPostalAddOnCode</td>
<td>[Optional Input] A string containing the postal add-on code. If the postal add-on code is not defined or not available, this parameter may be NULL.</td>
</tr>
<tr>
<td>pCoordPrecision</td>
<td>[Output] A pointer at a variable that receives a value indicating the precision of the computed coordinates, with numeric output values (see “Structures and Defines” on page 293, for specific values).</td>
</tr>
<tr>
<td>pLocation</td>
<td>[Output] A pointer to a structure that receives the geocoded coordinates.</td>
</tr>
<tr>
<td>pCandidateAddress</td>
<td>[Optional Output] A pointer to a the requested close address match. If the parameter is NULL, that information is not returned. If it is not NULL, the structure contains the input ZIP code, the census information (ZIP - state &amp; county FIPS, ZIP+2 census track, or ZIP+4-census block group) and the preferred last line for the input ZIP.</td>
</tr>
</tbody>
</table>

### Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading from database file.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>9</td>
<td>Error accessing database index.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_DATABASE_ERR</td>
<td>10</td>
<td>Invalid database contents.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_POSTAL_CODE</td>
<td>12</td>
<td>Invalid postal code.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENG_COORDS_NOTAVAILABLE</td>
<td>51</td>
<td>Postal centroid coordinates not available.</td>
</tr>
</tbody>
</table>
Structures and Defines

/*
 * Precision levels for Coordinates returned by the engine.
 */
#define NO_CENTROID 0
#define ZIPCODE_CENTROID 1
#define ZIPPLUS2_CENTROID 2
#define ZIPPLUS4_CENTROID 3
#define SHAPE_PATH_CENTER_COORDS 10
#define STREETADDRESS_COORDS 20
#define STREETINTERSECT_COORDS 30
#define POINTZIP 40
/

* DOUBLE_POINT structure definition. It is how we return all
* coordinates.
*/
typedef struct {
    double x;
    double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
/

* Defined constants for the addressType field in the
* CANDIDATE_ADDRESS structure.
*/
#define ADDRESS_TYPE_STREET 10
#define ADDRESS_TYPE_PLACE 11
#define ADDRESS_TYPE_ZIP 12
#define ADDRESS_TYPE_RURAL 13
#define ADDRESS_TYPE_HIGHWAY 14
#define ADDRESS_TYPE_POBOX 15
#define ADDRESS_TYPE_MILITARY 16
#define ADDRESS_TYPE_INTERSECTION 17
/

* The following values are OR’d together to produce a
* fieldsMatchedValue value in the CANDIDATE_ADDRESS structure.
*/
#define NON_CLOSE_MATCH 1
#define CLOSE_MATCH (1 << 1)
#define PREDIR_MATCH (1 << 2)
#define NAME_EXACT_MATCH (1 << 3)
#define POSTDIR_MATCH (1 << 4)
#define TYPE_MATCH (1 << 5)
#define POSTAL_CODE_MATCH (1 << 6)
#define HOUSE_NUMBER_MATCH (1 << 7)
#define USER_DICT_MATCH (1 << 8)
/

* Defined constants for the character array sizes in the ADDRESS
* structure.
*/
#define MAX_BUSINESS_LEN 256
#define MAXSTREET_LEN 256
#define MAX_CITY_LEN 40
#define MAXSTATE_LEN 40
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTALADDON_CODE_LEN 10
#define MAX_URBANIZATION_LEN 40
typedef struct {
char firm[MAX_BUSINESS_LEN];
char street[MAX_STREET_LEN];
char street2[MAX_STREET_LEN];
char city[MAX_CITY_LEN];
char state[MAX_STATE_LEN];
char postalCode[MAX_POSTAL_CODE_LEN];
char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
char urbanization[MAX_URBANIZATION_LEN];
} ADDRESS, *pADDRESS;
/*
* Defined constants for the character array sizes in the
* CANDIDATE_ADDRESS structure.
*/
#define MAX_CENSUS_BLOCK_LEN 20
#define MAX_STREET_LEN 256
#define MAX_LAST_LINE_LEN 40
#define MAX_CARRIER_ROUTE_LEN 5
#define DELIVERY_POINT_LEN 3
#define CHECK_DIGIT_LEN 2
#define MAX_LACS_LEN 2
#define PMB_RANGE_LEN 8
#define PMB_LEN 4

typedef struct
{
    short addressType;
    short fieldsMatchedValue;
    short score;
    BOOLEAN bPmbOutaddr2;
    ADDRESS candidateAddress;
    char censusBlockId[MAX_CENSUS_BLOCK_LEN];
    char tabbedAddress[MAX_STREET_LEN];
    char primaryStreet[MAX_STREET_LEN];
    char lastLine[MAX_LAST_LINE_LEN];
    char deliveryPoint[DELIVERY_POINT_LEN];
    char checkDigit[CHECK_DIGIT_LEN];
    char lacs[MAX_LACS_LEN];
    char carrierRoute[MAX_CARRIER_ROUTE_LEN];
    char record_type;
    char pmb_range[PMB_RANGE_LEN];
    char pmb[PMB_LEN];
} CANDIDATE_ADDRESS, *pCANDIDATE_ADDRESS;

For more information on the tabbedAddress field of the CANDIDATE_ADDRESS structure, see
tabbedAddress Field on page 350.

GeoEngGetAirport()

Purpose

This call returns airport location information. Use GeoEngGetAirport() by passing it an airport
location code. The call returns the location point and other information related to the airport.

Syntax

short GeoEngGetAirport(GEO_ENG_HANDLE geoEngHandle, 
char * pLoc_code,
pMEM_AIRPORT_REC pAirport_rec)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the</td>
</tr>
<tr>
<td></td>
<td>geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pLoc_code</td>
<td>[Input] Airport ID.</td>
</tr>
<tr>
<td>pAirport_rec</td>
<td>[Output] Pointer to structure to receive airport information.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_END_OF_DATA</td>
<td>11</td>
<td>No more data available from the requested set.</td>
</tr>
<tr>
<td>GEO_ENG_NO_APS</td>
<td>30</td>
<td>No airports found in given state.</td>
</tr>
<tr>
<td>GEO_ENG_LOW_LEVEL_AP_ERROR</td>
<td>31</td>
<td>Low level operation error.</td>
</tr>
</tbody>
</table>
Structures and Defines

#define LOC_ID_KEY_LEN 4
#define LOC_ID_LEN 4
#define AIRPORT_NAME_LEN 55
#define CFCC_LEN 3
#define USE_LEN 2
#define OWNERNAME_LEN 29
#define CONGES_LEV_LEN 1
#define SEVR_LEV_LEN 2
#define HUB_SIZE_LEN 1
#define TOWER_TYPE_LEN 1
#define STATE_LEN 2
#define STATE_KEY_LEN 2
#define FIPS_CNTY_LEN 5

typedef struct
{
    char locId[LOC_ID_LEN + 1];
    char airportName[AIRPORT_NAME_LEN + 1];
    char CFCC[CFCC_LEN + 1];
    char use[USE_LEN + 1];
    char ownerName[OWNERNAME_LEN + 1];
    char congesLev[CONGES_LEV_LEN + 1];
    char servLev[SEVR_LEV_LEN + 1];
    char hubSize[HUB_SIZE_LEN + 1];
    char towerType[TOWER_TYPE_LEN + 1];
    char state[STATE_LEN + 1];
    char FIPSCnty[FIPS_CNTY_LEN + 1];
    double X;
    double Y;
    int id;
    int elevation;
    int lrgCertEnp;
    int commEnp;
    int airTaxiEnp;
    int foreignEnp;
    int inTranEnp;
    int iRecNo;
} MEM_AIRPORT_REC, *pMEM_AIRPORT_REC;
GeoEngGetCandidateCoords()

Purpose
This call geocodes the user address that was previously supplied in the most recent call to GeoEngMatchAddress(), using the best possible approximating house number that was identified for a specific match candidate. The house number to be used is the same one as reported in the house number field in the candidate when accessed via GeoEngGetIndexedCandidate(). This function may invalidate pointers previously returned by GeoEngGetIndexedCandidate() or GeoEngGetNextHouseRange().

Syntax
```c
short GeoEngGetCandidateCoords(GEO_ENG_HANDLE geoEngHandle,
                               unsigned short candidateIndex,
                               short *pCoordPrecision,
                               pDOUBLE_POINT pLocation);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>candidateIndex</td>
<td>[Input] The zero based index of the match candidate, relative to the number of available candidates, as reported by GeoEngGetCandidateCount().</td>
</tr>
<tr>
<td>pCoordPrecision</td>
<td>[Output] A pointer at a variable that receives a value indicating the precision of the computed coordinates, with numeric output values (see “Structures and Defines” on page 298, for specific values).</td>
</tr>
<tr>
<td>pLocation</td>
<td>[Output] A pointer to a structure that receives the geocoded coordinates, which are computed using the current values of linearSetback and perpendicularInset.</td>
</tr>
</tbody>
</table>

Note: linearInset refers to the distance from the point to a street corner. This prevents the first house on two cross streets from displaying on top of one another. PerpendicularSetback is the distance from the center of the street. This allows points to display on the left and right sides of the street.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>Memory allocation failure.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>
/*
 * Precision levels for Coordinates returned by the engine.
 */
#define NO_CENTROID 0
#define ZIPCODE_CENTROID 1
#define ZIPPLUS2_CENTROID 2
#define ZIPPLUS4_CENTROID 3
#define SHAPE_PATH_CENTER_COORDS 10
#define STREET_ADDRESS_COORDS 20
#define STREET_INTERSECT_COORDS 30
#define POINT_ZIP 40
/
* DOUBLE_POINT structure definition. It is how we return all
* coordinates.
*/
typedef struct {
    double x;
    double y;
} DOUBLE_POINT, *pDOUBLE_POINT;

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading from database file.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>9</td>
<td>Error accessing database index.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_DATABASE_ERR</td>
<td>10</td>
<td>Invalid database contents.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_POSTAL_CODE</td>
<td>12</td>
<td>Invalid postal code.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENG_NO_DATA_AVAILABLE</td>
<td>15</td>
<td>Internal database inconsistency.</td>
</tr>
<tr>
<td>GEO_ENG_COORDS_NOTAVAILABLE</td>
<td>51</td>
<td>Postal centroid coordinates not available.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_CAND_INDEX</td>
<td>52</td>
<td>Match candidate index too large.</td>
</tr>
<tr>
<td>GEO_ENG_INTERP_HOUSE_ERR</td>
<td>819</td>
<td>Unable to interpolate house number.</td>
</tr>
<tr>
<td>GEO_ENG_INTERP_BAD_INSET</td>
<td>820</td>
<td>Invalid linear inset value.</td>
</tr>
<tr>
<td>GEO_ENG_INTERP_BAD_SETBACK</td>
<td>821</td>
<td>Invalid perpendicular setback value.</td>
</tr>
<tr>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>1075</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
</tbody>
</table>
GeoEngGetCandidateCount()

Purpose
This call returns the total number of candidates and/or the number of close match candidates as evaluated by the most recent call to GeoEngMatchAddress(). During matching, a weight is assigned to represent the result of matching against all of the component fields that make up the input and candidate addresses. The candidates are sorted in descending order of this assigned weight. This means that the candidates that are more similar to the input address have the lower candidate index values. A threshold value for match weight has been heuristically defined. If compared addresses have an assigned weight equal to or above this value, they are considered to be close matches.

Syntax
short GeoEngGetCandidateCount(GEO_ENG_HANDLE geoEngHandle,
                                 unsigned short *pCloseMatchCount,
                                 unsigned short *pCandidateCount);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the</td>
</tr>
<tr>
<td></td>
<td>geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pCloseMatchCount</td>
<td>[Output] A pointer at a variable that receives the number of close match</td>
</tr>
<tr>
<td></td>
<td>candidates.</td>
</tr>
<tr>
<td>pCandidateCount</td>
<td>[Output] A pointer at a variable that receives the number of candidates that</td>
</tr>
<tr>
<td></td>
<td>were evaluated by the most recent call to GeoEngMatchAddress() and that are</td>
</tr>
<tr>
<td></td>
<td>available for subsequent review.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>
GeoEngGetCandidateDpvInfo()

Purpose
This call returns the DPV information for a candidate.

Syntax
short GeoEngGetCandidateDpvInfo(GEO_ENG_HANDLE geoEngHandle,
Short candidateIndex,
pDPV_INFO pDpvInfo)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the</td>
</tr>
<tr>
<td></td>
<td>geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>candidateIndex</td>
<td>[Input] The zero-based index of the match candidate, relative to the number</td>
</tr>
<tr>
<td></td>
<td>of available candidates, as reported by GeoEngGetCandidateCount().</td>
</tr>
<tr>
<td>pDpvInfo</td>
<td>[Output] A pointer to a structure that receives the candidate’s DPV information.</td>
</tr>
</tbody>
</table>

Returns

Structures and Defines
#define MAX_FOOTNOTE_LEN 2
typedef struct {
char dpvCode
char cmra
char falsePositive
char footnote1[MAX_FOOTNOTE_LEN +1]
char footnote2[MAX_FOOTNOTE_LEN +1]
char footnote3[MAX_FOOTNOTE_LEN +1]
char footnote4[MAX_FOOTNOTE_LEN +1]
char footnote5[MAX_FOOTNOTE_LEN +1]
char footnote6[MAX_FOOTNOTE_LEN +1]
} DPV_INFO, *pDPV_INFO;

GeoEngGetCfgParms()

Purpose
This call gets current run time parameters for the geocoding engine.

Syntax
short GeoEngGetCfgParms(GEO_ENG_HANDLE geoEngHandle,
pGEO_ENG_CFG_PARMS pGeoEngParms);
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the</td>
</tr>
<tr>
<td></td>
<td>geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pGeoEngParams</td>
<td>[Output] A pointer to a structure that receives the current values of the</td>
</tr>
<tr>
<td></td>
<td>parameters that affect the behavior of the geocoding engine.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Keep in mind that if useCassMode is set to TRUE, you are only</td>
</tr>
<tr>
<td></td>
<td>using CASS rules. You are not actually CASS certified.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
</tbody>
</table>

Structures

/*
 * BOOLEAN with TRUE and FALSE.
 */
#define BOOLEAN short
/*
 * DIST_UNIT declaration and all the values it can have.
 */
typedef short DIST_UNIT, *pDIST_UNIT;
#define DIST_UNIT_FOOT 0
#define DIST_UNIT_DEGREE 1
#define DIST_UNIT_INCH 2
#define DIST_UNIT_LINK 3
#define DIST_UNIT_SURVEY_FOOT 4
#define DIST_UNIT_YARD 5
#define DIST_UNIT_ROD 6
#define DIST_UNIT_CHAIN 7
#define DIST_UNIT_MILE 8
#define DIST_UNIT NAUTICAL_MILE 9
#define DIST_UNIT_MILLIMETER 10
#define DIST_UNIT_CENTIMETER 11
#define DIST_UNIT_METER 12
#define DIST_UNIT_KILOMETER 13
typedef struct
{
    double linearInset;
    double perpendicularSetback;
    DIST_UNIT distUnit;
    BOOLEAN relaxHouseNumber;
    BOOLEAN relaxPostalCode;
    BOOLEAN relaxName;
    BOOLEAN relaxFinance;
    BOOLEAN relaxFinanceInState;
    BOOLEAN matchIntersections;
    BOOLEAN useCassMode;
    short relaxDistance;
    BOOLEAN exactCityName;
    BOOLEAN preferUserDictionary;
} GEO_ENG_CFG_PARMS, *pGEO_ENG_CFG_PARMS;

GeoEngGetDpvMode()

Purpose
This call returns the status of DPV mode: on or off.

Syntax
short GeoEngGetDpvMode(
    GEO_ENG_HANDLE geoEngHandle,  // input
    BOOLEAN *isDpvMode)                      // output

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>*isDpvMode</td>
<td>[Output] Indicates whether DPV mode is on or off: TRUE if DPV mode is on; FALSE if DPV mode is off.</td>
</tr>
</tbody>
</table>

Returns
The return value indicates whether DPV mode is on or off. TRUE indicates DPV mode is on; FALSE indicates that it’s off.

GeoEngGetErrorText()

Purpose
This call returns a description of a given error. Given a valid MapMarker error code, this method returns a string giving a short text description of the error. These text descriptions are listed in Appendix H: MapMarker GeoEngine Error Codes.

Syntax
short GeoEngGetErrorText(short errorCode, char* errorString)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorString</td>
<td>[Output] This is a pointer to a character array of no less than MAX_ERROR_STRING_SIZE. It is filled by the function with the text matching the given error code.</td>
</tr>
</tbody>
</table>

Returns

The return value indicates success or failure. If successful, the return code is SUCCESS. Otherwise one of the following is returned:

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>The error string pointer was invalid (NULL).</td>
</tr>
<tr>
<td>GEO_ENG_ERR_CODE_UNUSED</td>
<td>1113</td>
<td>The error code given does not match any known error in the database.</td>
</tr>
</tbody>
</table>

GeoEngGetIndexedCandidate()

Purpose

This call returns a specified match candidate that was evaluated by the most recent call to GeoEngMatchAddress(). The match candidates are ordered by decreasing similarity to the original user address. This allows access to candidates, regardless of their similarity to the user address. Each match candidate contains the best possible approximate house number for the user address in its house number field, considering all street house number ranges for that candidate. The user should not free the returned candidate pointer. This function may invalidate pointers previously returned by it or by GeoEngGetNextHouseRange().

Syntax

```c
short GeoEngGetIndexedCandidate(GEO_ENG_HANDLE geoEngHandle,
                               unsigned short candidateIndex,
                               pCANDIDATE_ADDRESS pCandidateAddress);
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>candidateIndex</td>
<td>[Input] The zero-based index of the match candidate desired, relative to the number of available candidates, as reported by GeoEngGetCandidateCount().</td>
</tr>
<tr>
<td>pCandidateAddress</td>
<td>[Output] A pointer to the requested close address match. The definition of this structure is given below.</td>
</tr>
</tbody>
</table>

### Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_DATABASE_ERR</td>
<td>10</td>
<td>Invalid database contents.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENG_NO_DATA_AVAILABLE</td>
<td>15</td>
<td>Internal database inconsistency.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_CAND_INDEX</td>
<td>52</td>
<td>Bad index number.</td>
</tr>
<tr>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>1075</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
</tbody>
</table>
Structures and Defines

/*
 * Defined constants for the addressType field in the
 * CANDIDATE_ADDRESS structure.
 */
#define ADDRESS_TYPE_STREET 10
#define ADDRESS_TYPE_PLACE 11
#define ADDRESS_TYPE_ZIP 12
#define ADDRESS_TYPE_RURAL 13
#define ADDRESS_TYPE_HIGHWAY 14
#define ADDRESS_TYPE_POBOX 15
#define ADDRESS_TYPE_MILITARY 16
#define ADDRESS_TYPE_INTERSECTION 17

/*
 * The following values are OR’d together to produce a
 * fieldsMatchedValue value in the CANDIDATE_ADDRESS structure.
 */
#define NON_CLOSE_MATCH 1
#define CLOSE_MATCH (1 << 1)
#define PREDIR_MATCH (1 << 2)
#define NAME_EXACT_MATCH (1 << 3)
#define POSTDIR_MATCH (1 << 4)
#define TYPE_MATCH (1 << 5)
#define POSTAL_CODE_MATCH (1 << 6)
#define HOUSE_NUMBER_MATCH (1 << 7)
#define USER_DICT_MATCH (1 << 8)
#define CITY_MATCH (1 << 9)
#define INTERSECT_STREET1_UD (1 << 10)
#define INTERSECT_STREET2_UD (1 << 11)
#define FIRM_MATCH (1 << 12)

/*
 * Defined constants for the character array sizes in the ADDRESS
 * structure.
 */
#define MAX_BUSINESS_LEN 256
#define MAX_STREET_LEN 256
#define MAX_CITY_LEN 40
#define MAX_STATE_LEN 40
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10
#define MAX_URBANIZATION_LEN 40
typedef struct {
    char firm[MAX_BUSINESS_LEN];
    char street[MAX_STREET_LEN];
    char street2[MAX_STREET_LEN];
    char city[MAX_CITY_LEN];
    char state[MAX_STATE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
    char urbanization[MAX_URBANIZATION_LEN];
} ADDRESS, *pADDRESS;

/*
* Defined constants for the character array sizes in the
* CANDIDATE_ADDRESS structure.
*/
#define MAX_CENSUS_BLOCK_LEN 20
#define MAX_STREET_LEN 256
#define MAX_LAST_LINE_LEN 40
#define MAX_CARRIER_ROUTE_LEN 5
#define DELIVERY_POINT_LEN 3
#define CHECK_DIGIT_LEN 2
#define MAX_LACS_LEN 2

typedef struct {
    short addressType;
    short fieldsMatchedValue;
    short score;
    BOOLEAN bPmbOutaddr2;
    ADDRESS candidateAddress;
    char censusBlockId[MAX_CENSUS_BLOCK_LEN];
    char tabbedAddress[MAX_STREET_LEN];
    char primaryStreet[MAX_STREET_LEN];
    char lastLine[MAX_LAST_LINE_LEN];
    char deliveryPoint[DELIVERY_POINT_LEN];
    char checkDigit[CHECK_DIGIT_LEN];
    char lacs[MAX_LACS_LEN];
    char carrierRoute[MAX_CARRIER_ROUTE_LEN];
    char record_type;
    char pmb_range[PMB_RANGE_LEN];
    char pmb[PMB_LEN];
} CANDIDATE_ADDRESS, *pCANDIDATE_ADDRESS;

For more information on the tabbedAddress field, see tabbedAddress Field on page 350 in the Data Structures appendix.

GeoEngGetIndexedCoords()

Purpose

This call geocodes a user address that was previously supplied in the most recent call to GeoEngMatchAddress(), using a specific street house number range for a selected match candidate. This function may invalidate pointers previously returned by GeoEngGetIndexedCandidate() or GeoEngGetNextHouseRange().
Syntax

```c
short GeoEngGetIndexedCoords(GEO_ENG_HANDLE geoEngHandle,
   unsigned short candidateIndex,
   unsigned short rangeIndex,
   short *pCoordPrecision,
   pDOUBLE_POINT pLocation);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>candidateIndex</td>
<td>[Input] The zero based index of the match candidate, relative to the number of available candidates, as reported by GeoEngGetCandidateCount().</td>
</tr>
<tr>
<td>rangeIndex</td>
<td>[Input] The zero based index of the street house number range, relative to the full set of street house number ranges for the specified match candidate.</td>
</tr>
<tr>
<td>pCoordPrecision</td>
<td>[Output] A pointer at a variable that receives a value indicating the precision of the computed coordinates, with numeric output values (see Structures and Defines section on page 308, for specific values).</td>
</tr>
<tr>
<td>pLocation</td>
<td>[Output] A pointer to a structure that receives the geocoded coordinates, which are computed using the current values of linearSetback and perpendicularInset.</td>
</tr>
</tbody>
</table>

**Note:** linearInset refers to the distance from the point to a street corner. This prevents the first house on two cross streets from displaying on top of one another. PerpendicularSetback is the distance from the center of the street. This allows points to display on the left and right sides of the street.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>Memory allocation failure.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_DATABASE_ERR</td>
<td>10</td>
<td>Invalid database contents.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_POSTAL_CODE</td>
<td>12</td>
<td>Invalid postal code.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENG_NO_DATA_AVAILABLE</td>
<td>15</td>
<td>Internal database inconsistency.</td>
</tr>
</tbody>
</table>
Structures and Defines

/* Precision levels for Coordinates returned by the engine. */
#define NO_CENTROID 0
#define ZIPCODE_CENTROID 1
#define ZIPPLUS2_CENTROID 2
#define ZIPPLUS4_CENTROID 3
#define SHAPE_PATH_CENTER_COORDS 10
#define STREET_ADDRESS_COORDS 20
#define STREET_INTERSECT_COORDS 30
#define POINT_ZIP 40

/* DOUBLE_POINT structure definition. It is how we return all coordinates. */
typedef struct {
    double x;
    double y;
} DOUBLE_POINT, *pDOUBLE_POINT;

GeoEngGetIndexedHwyExit()

Purpose
This function returns the highway name, exit number, and/or exit name, as well as the ZIP Code, city, and coordinates of the exit location of the matched candidate. This method uses a zero-based index number to retrieve the candidate. The first candidate is 0.

This function cannot be used until its corresponding call, GeoEngMatchHwyExit(), has been invoked. Using the count from that function, the individual candidates can be retrieved from GeocodeGetHwyExitCandidate().

Syntax

short GeoEngGetIndexedHwyExit(GEO_ENG_HANDLE geoEngHandle,
    pHWYEXIT_RECORD pHwyExitRec,
    short index)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pHwyExitRec</td>
<td>[Output] This structure holds all the returned information on the exit, including prefix type, highway name, suffix type, direction, exit number, exit name, city, state, ZIP Code, and the X and Y coordinates.</td>
</tr>
<tr>
<td>index</td>
<td>[Input] Zero-based candidate index to indicate the candidate in the array that should be returned.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>

Structures and Defines

```c
#define EXIT_ID_LEN 10
#define HWY_PRETYPE_LEN 20
#define HWY_NAME_LEN 40
#define HWY_POSTTYPE_LEN 20
#define HWYDIR_LEN 2
#define EXIT_NUM_LEN 10
#define CITY_LEN 40
#define STATE_LEN 2
#define ZIPCODE_LEN 5

typedef struct
{
  char exitID(EXIT_ID_LEN+1);
  char hwyTypePre(HWY_PRETYPE_LEN+1);
  char hwyName(HWY_NAME_LEN+1);
  char hwyTypeSuf(HWY_POSTTYPE_LEN+1);
  char exitNum(EXIT_NUM_LEN+1);
  char exitName(EXIT_NAME_LEN+1);
  char city(CITY_LEN+1);
  char state(STATE_LEN+1);
  char zipcode(ZIPCODE_LEN+1);
  double X;
  double Y;
} HWYEXIT_RECORD, *pHWYEXIT_RECORD;
```
GeoEngGetNextHouseRange()

Purpose
This call returns the next street house range for the specified match candidate from the candidates that were evaluated by the most recent call to GeoEngMatchAddress(). The user should not free the returned house range pointer. This function overwrites the structure referenced by any previous calls to it. If the first call to this function for a given match candidate does not specify firstRange equal to TRUE, results are unpredictable, and an error may be returned.

Syntax

```c
short GeoEngGetNextHouseRange(GEO_ENG_HANDLE geoEngHandle,
                                unsigned short candidateIndex,
                                BOOLEAN firstRange,
                                pHOUSE_RANGE pHouseRange);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>candidateIndex</td>
<td>[Input] The zero based index of the match candidate, relative to the number of available candidates, as reported by GeoEngGetCandidateCount().</td>
</tr>
<tr>
<td>firstRange</td>
<td>[Input] TRUE if the first house number range for the match candidate is desired; FALSE if the next range is to retrieved, in terms of sequentially stepping through all the house number ranges for the street.</td>
</tr>
<tr>
<td>pHouseRange</td>
<td>[Output] A variable that receives a pointer to the requested street house number range.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>10</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
<tr>
<td>GEO_ENG_END_OF_DATA</td>
<td>11</td>
<td>No more data available from the requested set.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
</tbody>
</table>
/**
 * Defined constants for the character array sizes in all the
 * HOUSE_RANGE structure.
 */
#define MAX_HOUSE_NUMBER_LEN 25
#define MAX_UNIT_TYPE_LEN 5
#define MAX_UNIT_VALUE_LEN 9
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10
#define MAX_PLACE_NAME_LEN 35

typedef struct
{
    HOUSE_PARITY houseParity;
    char fromHouse[MAX_HOUSE_NUMBER_LEN];
    char toHouse[MAX_HOUSE_NUMBER_LEN];
    char unitType[MAX_UNIT_TYPE_LEN];
    char fromUnit[MAX_UNIT_VALUE_LEN];
    char toUnit[MAX_UNIT_VALUE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
    char placeName[MAX_PLACE_NAME_LEN];
    char lacs;
    char carrRoute[MAX_CARRIER_ROUTE_LEN];
} HOUSE_RANGE, *pHOUSE_RANGE;

GeoEngGetStatesFound()

Purpose
This call returns a space-separated list of 2-digit state abbreviations of the states in the Address
Dictionary path.

Syntax
short GeoEngGetStatesFound(GEO_ENG_HANDLE geoEngHandle,
                            char *stateAbbList);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by</td>
</tr>
<tr>
<td></td>
<td>the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>stateAbbList</td>
<td>[Output] A list of 2-digit state abbreviations separated by spaces that</td>
</tr>
<tr>
<td></td>
<td>define the states found in the Address Dictionary path.</td>
</tr>
</tbody>
</table>
GeoEngGetStatesLicensed()

Purpose
This call gets licensed states. GeoEngGetStatesLicensed() returns a space-separated list of 2-digit state abbreviations for the licensed states.

Syntax
```
short GeoEngGetStatesLicensed(GEO_ENG_HANDLE geoEngHandle,
                               char *stateAbbList);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the</td>
</tr>
<tr>
<td></td>
<td>geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>stateAbbList</td>
<td>[Output] A list of 2-digit state abbreviations separated by spaces that</td>
</tr>
<tr>
<td></td>
<td>define the states found in the Address Dictionary path.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>1075</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
</tbody>
</table>
**GeoEngGetVersion()**

**Purpose**

This call returns the engine version number. For MapMarker 10.x, GeoEngGetVersion() returns the value of “10.0.”

**Syntax**

```
short GeoEngGetVersion(double *pVersionNum);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>

**GeoEngGetZIPCountyFIPSList()**

**Purpose**

This call returns an array of structures, which contain the FIPS codes and names of the counties that contain the input ZIP Code.

**Syntax**

```
short GeoEngGetZIPCountyFIPSList(GEO_ENG_HANDLE geoEngHandle,
char *zip,
pCOUNTY_FIPS_LIST pCountyFIPSList);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>zip</td>
<td>[Input] ZIP Code to find</td>
</tr>
<tr>
<td>pCountyFIPSList</td>
<td>[Output] The counties and FIPS Codes for the input ZIP Code.</td>
</tr>
</tbody>
</table>
Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>1075</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
</tbody>
</table>

Structures

typedef struct
{
    Long numCountyFips;
    CNTY_NAME_FIPS_REC countyFipsList[MAX_NUM_COUNTIES];
} COUNTY_FIPS_LIST, *pCOUNTY_FIPS_LIST;

typedef struct
{
    char fipsCode[MAX_FIPS_LEN + 1]
    char cntyName[MAX_CNTY_NAME_LEN + 1];
} CNTY_NAME_FIPS_REC, *pCNTY_NAME_FIPS_REC;

GeoEngInit()

Purpose
This call initializes the Geocoding Engine. It initializes all subsystems that are required to support the capabilities of the geocoding engine.

Syntax

short GeoEngInit(char *pSearchPath,
char *pDbName,
char *pUserDictionaryPath,
short dbInitRequestFlags,
pGEO_ENG_HANDLE pGeoEngHandle);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSearchPath</td>
<td>[Input] Directory search path to locate the database and configuration files. This is a string containing directory paths separated by semicolons.</td>
</tr>
<tr>
<td>pDbName</td>
<td>[Input] String containing the base name to use in constructing the names of the database and configuration files.</td>
</tr>
<tr>
<td>pUserDictionaryPath</td>
<td>[Input] Path to the user dictionary files.</td>
</tr>
</tbody>
</table>
Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbInitRequestFlags</td>
<td>[Output] A pointer to an integer that receives a value indicating which</td>
</tr>
<tr>
<td></td>
<td>dictionary components were found. The value is the bitwise OR of the</td>
</tr>
<tr>
<td></td>
<td>following defined constants and indicates which dictionary components can</td>
</tr>
<tr>
<td></td>
<td>be initialized using the specified search path and Address Dictionary name.</td>
</tr>
<tr>
<td></td>
<td>• 1 – STREET_DB</td>
</tr>
<tr>
<td></td>
<td>• 2 – CENTROID_DB</td>
</tr>
<tr>
<td></td>
<td>• 3 – STREET_DB and CENTROID_DB</td>
</tr>
<tr>
<td></td>
<td>• 4 – USER_DB</td>
</tr>
<tr>
<td></td>
<td>• 5 – STREET_DB and USER_DB</td>
</tr>
<tr>
<td></td>
<td>• 6 – CENTROID_DB and USER_DB</td>
</tr>
<tr>
<td></td>
<td>• 7 – STREET_DB, CENTROID_DB, and USER_DB</td>
</tr>
<tr>
<td>pGeoEngHandle</td>
<td>[Output] A pointer to an opaque handle that references a structure used</td>
</tr>
<tr>
<td></td>
<td>internally by the geocoding engine.</td>
</tr>
</tbody>
</table>

Return # Description

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function.</td>
</tr>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>Memory allocation of internal structure failed.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_NOT_FOUND_ERR</td>
<td>4</td>
<td>Input file was not found.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_OPEN_ERR</td>
<td>5</td>
<td>Error opening sinames, cityfin, usafin, citylin, file.tbl, license file, etc.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading sinames, cityfin, usafin, citylin, file.tbl, license file, etc.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_OPEN_ERR</td>
<td>8</td>
<td>Error while trying to open a dictionary index.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>9</td>
<td>Error while trying to read a dictionary index.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_LICFILE_ERR</td>
<td>18</td>
<td>Invalid license file.</td>
</tr>
<tr>
<td>GEO_ENG_MULTIPLE_INSTANCE</td>
<td>58</td>
<td>Initialize has been called multiple times.</td>
</tr>
<tr>
<td>GEO_ENG_PARSE_INIT_ERR</td>
<td>307</td>
<td>Parsing file (geo_usa.*) missing or bad.</td>
</tr>
<tr>
<td>Return</td>
<td>Return #</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GEO_ENG_SINAMES_INDEX_OPEN</td>
<td>1049</td>
<td>Could not open sinames.idx.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_DB_INIT_FLAGS</td>
<td>1076</td>
<td>Invalid dbTypes specified (valid are STREET_DB, CENTROID_DB, and USER_DB.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_LICFILE</td>
<td>1079</td>
<td>License file not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_FILEID_TABLE</td>
<td>1080</td>
<td>File.tbl not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_ADR_ADX</td>
<td>1081</td>
<td>.adr/.adx files not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_BAD_HEADER</td>
<td>1082</td>
<td>Unable to read the header of file.tbl.</td>
</tr>
<tr>
<td>GEO_ENG_BETA_EXPIRATION_ERR</td>
<td>1084</td>
<td>Beta has expired.</td>
</tr>
<tr>
<td>GEO_ENG_INVALID_SERIAL_NUMBER</td>
<td>1085</td>
<td>Invalid serial number.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_USER_DICT_INIT_ERR</td>
<td>1087</td>
<td>Error finding user dictionary files.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_FILE</td>
<td>1089</td>
<td>cityline.idx not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_OPEN</td>
<td>1090</td>
<td>Could not open cityline.idx.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_LICFILE_EXPIRED</td>
<td>1091</td>
<td>Trial license has expired.</td>
</tr>
<tr>
<td>GEO_ENG_MAX_USER_DICT_EXCEEDED_ERR</td>
<td>1092</td>
<td>Exceeded limit on number of user dictionaries allowed.</td>
</tr>
<tr>
<td>GEO_ENG_SINAMES_INDEX_FILE</td>
<td>1093</td>
<td>Could not find sinames.idx.</td>
</tr>
<tr>
<td>GEO_ENG_Z9_CEN_FILE_NOT_FOUND_ERR</td>
<td>1095</td>
<td>ZIP centroid (.cen) files not found.</td>
</tr>
<tr>
<td>GEO_ENG_UNIQUE_ZIP_INDEX_FILE</td>
<td>1097</td>
<td>Could not find cityunz.idx.</td>
</tr>
<tr>
<td>GEO_ENG_UNIQUE_ZIP_INDEX_OPEN</td>
<td>1098</td>
<td>Could not open cityunz.idx.</td>
</tr>
<tr>
<td>GEO_ENG_URBANIZ_ZIP_INDEX_FILE</td>
<td>1101</td>
<td>Could not find cityurb.idx.</td>
</tr>
<tr>
<td>GEO_ENG_URBANIZ_ZIP_INDEX_OPEN</td>
<td>1102</td>
<td>Could not open cityurb.idx.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_NOT_FOUND_ERR</td>
<td>1228</td>
<td>Zipmastr.cdb or Zipmastr.jdx not found.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_READ_ERR</td>
<td>1230</td>
<td>Error reading the Zipmastr file.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_INDEX_OPEN_ERR</td>
<td>1232</td>
<td>Error opening the Zipmastr.idx file.</td>
</tr>
<tr>
<td>GEO_ENG_NADCON_MISSING_FILE</td>
<td>1334</td>
<td><em>.las/</em>.los files not found.</td>
</tr>
</tbody>
</table>
GeoEngInitWithLicense()

Purpose
This call initializes the Geocoding Engine when the correct serial number and license file is supplied.

Syntax

```c
short GeoEngInitWithLicense(char *pSearchPath,
char *pDbName,
char *pUserDictPath,
short dbInitRequestFlags,
pGEO_ENG_HANDLE pGeoEngHandle)
char *pSerialNumber,
char licenseExt[4];
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSearchPath</td>
<td>[Input] Directory search path to locate the database and configuration files. This is a string containing directory paths separated by semicolons.</td>
</tr>
<tr>
<td>pDbName</td>
<td>[Input] String containing the base name to use in constructing the names of the database and configuration files.</td>
</tr>
<tr>
<td>pUserDictPath</td>
<td>[Input] Path to the user dictionary files</td>
</tr>
</tbody>
</table>
| dbInitRequestFlags | [Input] An integer that indicates which database components to initialize. The value is the bitwise OR of the following defined constants and indicates which database components are requested to be initialized using the specified search path and database name.  
  • 1 – STREET_DB  
  • 2 – CENTROID_DB  
  • 3 – STREET_DB and CENTROID_DB  
  • 4 – USER_DB  
  • 5 – STREET_DB and USER_DB  
  • 6 – CENTROID_DB and USER_DB  
  • 7 – STREET_DB, CENTROID_DB, and USER_DB |
| pgeoEngHandle      | [Output] A pointer to an opaque handle that references a structure used internally by the geocoding engine. |
| pSerialNumber      | [Input] Serial number to use.                                               |
| licenseExt         | [Input] License file extension.                                             |
# Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function.</td>
</tr>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>Memory allocation of internal structure failed.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_NOT_FOUND_ERR</td>
<td>4</td>
<td>Input file was not found.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_OPEN_ERR</td>
<td>5</td>
<td>Error opening sinames, cityfin, usafin, citylin, file.tbl, license file, etc.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading sinames, cityfin, usafin, citylin, file.tbl, license file, etc.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_OPEN_ERR</td>
<td>8</td>
<td>Error while trying to open a dictionary index.</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>9</td>
<td>Error while trying to read a dictionary index.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_LICFILE_ERR</td>
<td>18</td>
<td>Invalid license file.</td>
</tr>
<tr>
<td>GEO_ENG_MULTIPLE_INSTANCE</td>
<td>58</td>
<td>Initialize has been called multiple times.</td>
</tr>
<tr>
<td>GEO_ENG_PARSE_INIT_ERR</td>
<td>307</td>
<td>Parsing file (geo_usa.*) missing or bad.</td>
</tr>
<tr>
<td>GEO_ENG_SINAMES_INDEX_OPEN</td>
<td>1049</td>
<td>Could not open sinames.idx.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_DB_INIT_FLAGS</td>
<td>1076</td>
<td>Invalid dbTypes specified (valid are STREET_DB, CENTROID_DB, and USER_DB.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_LICFILE</td>
<td>1079</td>
<td>License file not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_FILEID_TABLE</td>
<td>1080</td>
<td>File.tbl not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_ADR_ADX</td>
<td>1081</td>
<td>.adr/.adx files not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_BAD_HEADER</td>
<td>1082</td>
<td>Unable to read the header of File.tbl.</td>
</tr>
<tr>
<td>GEO_ENG_BETA_EXPIRATION_ERR</td>
<td>1084</td>
<td>Beta has expired.</td>
</tr>
<tr>
<td>GEO_ENG_INVALID_SERIAL_NUMBER</td>
<td>1085</td>
<td>Invalid serial number.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_USER_DICT_INIT_ERR</td>
<td>1087</td>
<td>Error finding user dictionary files.</td>
</tr>
</tbody>
</table>
GeoEngIsCandidateMultiUnit()

`GeoEngIsCandidateMultiUnit()` determines if a specified candidate address in the candidate array contains multiple units. This call checks to see if a specified candidate address has multiple units associated with it, such as apartment or suite numbers in a building. If `GeoEngIsCandidateMultiUnit()` is called when the number of candidates is 0, the function may access a candidate list from a previous match that is no longer valid. The function should only be called after a call to `GeoEngGetCandidateCount()` which determines whether a candidate list exists for the address in question.

`GeoEngIsCandidateMultiUnit()` can be called after a candidate list has been generated. It is not necessary to have retrieved the candidate via `GeoEngGetIndexedCandidate()`. In order for a candidate to be evaluated by `GeoEngIsCandidateMultiUnit()`, the candidate must have a 4-digit ZIP Code extension (ZIP+4). If the candidate does not have the ZIP+4, the function returns an error 15 (GEO_ENG_NO_DATA_AVAILABLE).

`GeoEngIsCandidateMultiUnit()` cannot evaluate intersection candidates since they do not return candidate information, only a point. Intersection candidates always return SUCCESS with `IsMultiUnit` set to FALSE.

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_FILE</td>
<td>1089</td>
<td>cityline.idx not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_OPEN</td>
<td>1090</td>
<td>Could not open cityline.idx.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_LICFILE_EXPIRED</td>
<td>1091</td>
<td>Trial license has expired.</td>
</tr>
<tr>
<td>GEO_ENG_MAX_USER_DICT_EXCEEDED_ERR</td>
<td>1092</td>
<td>Exceeded limit on number of user dictionaries allowed.</td>
</tr>
<tr>
<td>GEO_ENG_SINAMES_INDEX_FILE</td>
<td>1093</td>
<td>Could not find sinames.idx.</td>
</tr>
<tr>
<td>GEO_ENG_Z9_CEN_FILE_NOT_FOUND_ERR</td>
<td>1095</td>
<td>ZIP centroid (.cen) files not found.</td>
</tr>
<tr>
<td>GEO_ENG_UNIQUE_ZIP_INDEX_FILE</td>
<td>1097</td>
<td>Could not find cityunz.idx.</td>
</tr>
<tr>
<td>GEO_ENG_UNIQUE_ZIP_INDEX_OPEN</td>
<td>1098</td>
<td>Could not open cityunz.idx.</td>
</tr>
<tr>
<td>GEO_ENG_URBANIZ ZIP_INDEX_FILE</td>
<td>1101</td>
<td>Could not find cityurb.idx.</td>
</tr>
<tr>
<td>GEO_ENG_URBANIZ ZIP_INDEX_OPEN</td>
<td>1102</td>
<td>Could not open cityurb.idx.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_NOT_FOUND_ERR</td>
<td>1228</td>
<td>Zipmastr.cdb or Zipmastr.jdx not found.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_READ_ERR</td>
<td>1230</td>
<td>Error reading the Zipmastr file.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_INDEX_OPEN_ERR</td>
<td>1232</td>
<td>Error opening the Zipmastr.idx file.</td>
</tr>
<tr>
<td>GEO_ENG_NADCON_MISSING_FILE</td>
<td>1334</td>
<td><em>.las/</em>.los files not found.</td>
</tr>
</tbody>
</table>

- Return # Description
Syntax

short GeoEngIsCandidateMultiUnit(GEO_ENG_HANDLE geoEngHandle,
short candidateIndex,
BOOLEAN * IsMultiUnit)

Parameters

Parameter | Description
---|---
geoEngHandle | [Input] Opaque handle that references an internal engine structure.
candidateIndex | [Input] Zero-based index of candidate to examine.
IsMultiUnit | [Output] Boolean Flag indicating whether the candidate address is multi-unit.

Returns

Return | Return # | Description
---|---|---
SUCCESS | 0 | Normal return for the function.
GEO_ENG_BAD_CAND_INDEX | 52 | Match candidate index too large.
GEO_ENG_BAD_PARAM_ERR | 3 | NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR | 2 | Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR | 10 | Error accessing the Address Dictionary.
GEO_ENG_INVALID_ACCESS_ERR | 16 | Invalid attempt to access data.
GEO_ENG_NO_DATA_AVAILABLE | 15 | Internal database inconsistency.
GEO_ENG_UNINIT_DB_COMPONENT | 13 | Requested database component failed to initialize.

GeoEngIsEval()

Purpose

This call determines if the engine has been initialized for an evaluation copy of the Address Dictionary.

Syntax

short GeoEngIsEval(GEO_ENG_HANDLE geoEngHandle,
BOOLEAN *bEval);
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>bEval</td>
<td>[Output] A pointer to a BOOLEAN value which is returned as TRUE if it is an evaluation copy.</td>
</tr>
</tbody>
</table>

## Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_EXCEEDED_LIMIT</td>
<td>19</td>
<td>Exceeded trial license limit.</td>
</tr>
</tbody>
</table>

## GeoEngKill()

### Purpose
This call shuts down the Geocoding Engine. GeoEngKill() cleans up after the geocoding engine driver and the subsystems that were also initialized in response to a previous call to GeoEngInit().

### Syntax
```
short GeoEngKill(pGEO_ENG_HANDLE pGeoEngHandle);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pGeoEngHandle</td>
<td>[Input/Output] A pointer to an opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
</tbody>
</table>

## Return

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>
GeoEngMatchAddress()

Purpose
This call matches a user address against the Address Dictionary. GeoEngMatchAddress() accepts a user address that is already broken down into street address, city, state, and postal code. This call parses this address into a standard form, and then matches it against the Address Dictionary. An optional output parameter allows the user to view the parsed form of the address. For efficient use of the MapMarker subsystems, a file of input addresses should be sorted on postal code before this function is invoked. The user should not attempt to free the parsed address pointer. The geocoding engine supports access to match candidates for only one input address at a time. This call destroys all information related to previous calls to this function for different addresses, and it invalidates the previous values of assigned pointers returned by this function, GeoEngGetIndexedCandidate(), or GeoEngGetNextHouseRange(). This function can be used only if STREET_DB is initialized. After this call, geocoding can be done using any of the match candidates via the functions GeoEngGetCandidateCoords() or GeoEngGetIndexedCoords().

Syntax

```
short GeoEngMatchAddress(GEO_ENG_HANDLE geoEngHandle,
pADDRESS pInputAddress
pPARSED_ADDRESS pParsedAddress
pMATCH_STATUS pMatchStatus);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pInputAddress</td>
<td>[Input] A pointer to a structure that contains the input address.</td>
</tr>
<tr>
<td>pParsedAddress</td>
<td>[Optional Output] A pointer to a structure that contains the cleaned (parsed) input street address. If the cleaned street address is not desired, pParsedAddress may be NULL.</td>
</tr>
<tr>
<td>pMatchStatus</td>
<td>[Output] A pointer to a variable that receives the match status, which is an enumeration type defined later in this document (see “Structures and Defines” on page 324, for specific values).</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>Memory allocation of internal structure failed.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>Return</td>
<td>Return #</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GEO_ENGINE_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENGINE_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading from finance record from database file.</td>
</tr>
<tr>
<td>GEO_ENGINE_FILE_WRITE_ERR</td>
<td>7</td>
<td>Couldn’t close previous .adx.</td>
</tr>
<tr>
<td>GEO_ENGINE_INDEX_OPEN_ERR</td>
<td>8</td>
<td>Couldn’t open next .adx.</td>
</tr>
<tr>
<td>GEO_ENGINE_END_OF_DATA</td>
<td>11</td>
<td>Premature end of data found.</td>
</tr>
<tr>
<td>GEO_ENGINE_BAD_POSTAL_CODE</td>
<td>12</td>
<td>Postal code is not valid and city/state is not valid.</td>
</tr>
<tr>
<td>GEO_ENGINE_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENGINE_BAD_INPUT_ADDRESS</td>
<td>14</td>
<td>Input address is invalid or missing fields.</td>
</tr>
<tr>
<td>GEO_ENGINE_NO_DATA_AVAILABLE</td>
<td>15</td>
<td>Internal database inconsistency.</td>
</tr>
<tr>
<td>GEO_ENGINE_BAD_INTERSECT_FORMAT</td>
<td>59</td>
<td>Couldn’t find street or intersection in input.</td>
</tr>
<tr>
<td>GEO_ENGINE_PARSE_ENGINE_ERR</td>
<td>308</td>
<td>Parsing subsystem error.</td>
</tr>
<tr>
<td>GEO_ENGINE_NO_STREET_ADDRESS</td>
<td>309</td>
<td>Street address was missing.</td>
</tr>
<tr>
<td>GEO_ENGINE_PARSE_STACK_OVERFLOW</td>
<td>310</td>
<td>Parser stack overflow (probably bad address).</td>
</tr>
<tr>
<td>GEO_ENGINE_DATABASE_ACCESS_ERR</td>
<td>1075</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
<tr>
<td>GEO_ENGINE_UNDEFINED_SEARCH</td>
<td>1077</td>
<td>Trying to get next street without setting up the first.</td>
</tr>
<tr>
<td>GEO_ENGINE_MISSING_DATABASE_ERR</td>
<td>1078</td>
<td>Data for street not licensed or not found.</td>
</tr>
<tr>
<td>GEO_ENGINE_DBQ_UDRADR_CONFLICT_ERR</td>
<td>1086</td>
<td>Internal problem with flags defining user dictionary or address dictionary searching.</td>
</tr>
</tbody>
</table>
Structures and Defines

/*
 * Defined constants for the character array sizes in the ADDRESS
 * structure.
 */
#define MAX_BUSINESS_LEN 256
#define MAX_STREET_LEN 256
#define MAX_CITY_LEN 40
#define MAX_STATE_LEN 40
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10
#define MAX_URBANIZATION_LEN 40

typedef struct
{
    char firm[MAX_BUSINESS_LEN];
    char street[MAX_STREET_LEN];
    char street2[MAX_STREET_LEN];
    char city[MAX_CITY_LEN];
    char state[MAX_STATE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
    char urbanization[MAX_URBANIZATION_LEN];
} ADDRESS, *pADDRESS;

/*
 * Defined constants for the character array sizes in the
 * PARSED_ADDRESS structure.
 */
#define MAX_HOUSE_NUMBER_LEN 25
#define MAX_HOUSE_PREFIX_LEN 11
#define MAX_HOUSE_SEPARATOR_LEN 3
#define MAX_HOUSE_COORD_LEN 8
#define MAX_HOUSE_SUFFIX_LEN 11
#define MAX_STREET_PREDIR_LEN 3
#define MAX_STREET_PRETYPE_LEN 11
#define MAX_STREET_NAME_LEN 29
#define MAX_STREET_POSTTYPE_LEN 5
#define MAX_STREET_POSTDIR_LEN 3
#define MAX_POSTBOX_LEN 8
#define MAX_UNIT_TYPE_LEN 5
#define MAX_UNIT_VALUE_LEN 9
typedef struct {
    char housePrefix[MAX_HOUSE_PREFIX_LEN];
    char houseNumber[MAX_HOUSE_NUMBER_LEN];
    char houseSeparator[MAX_HOUSE_SEPARATOR_LEN];
    char houseCoord[MAX_HOUSE_COORD_LEN];
    char houseSuffix[MAX_HOUSE_SUFFIX_LEN];
    char preDir[MAX_STREET_PREDIR_LEN];
    char preType[MAX_STREET_PRETYPE_LEN];
    char name[MAX_STREET_NAME_LEN];
    char postType[MAX_STREET_POSTTYPE_LEN];
    char postDir[MAX_STREET_POSTDIR_LEN];
    char postalBox[MAX_POSTBOX_LEN];
    /* no longer used in version 3 */
    char unitType[MAX_UNIT_TYPE_LEN];
    char unitValue[MAX_UNIT_VALUE_LEN];
} PARSED_ADDRESS, *pPARSED_ADDRESS;

/*
 * MATCH_STATUS declaration, as well as the Defined constants for
 * the values it can be.
*/
typedef short MATCH_STATUS, *pMATCH_STATUS;
#define INTERSECT_BASE 4
#define SINGLE_MATCH 0
#define MULTIPLE_MATCHES 1
#define NO_MATCHES 2
#define NO_CANDIDATES 3
#define SINGLE_INTERSECT_MATCH 0 + INTERSECT_BASE
#define MULTIPLE_INTERSECT_MATCH 1 + INTERSECT_BASE
#define NO_INTERSECT_MATCHES 2 + INTERSECT_BASE
#define NO_INTERSECT_CANDIDATES 3 + INTERSECT_BASE
#define POSSIBLE_INTERSECTION 4 + INTERSECT_BASE

GeoEngMatchFormattedAddress()

Purpose
This call matches a user address that is already broken down into street address and last line components against the Address Dictionary.

Syntax
short GeoEngMatchFormattedAddress(GEO_ENG_HANDLE geoEngHandle,
    pPARSED_ADDRESS pParsedAddress1,
    pPARSED_ADDRESS pParsedAddress2,
    pLAST_LINE pLastLine,
    pMATCH_STATUS pMatchStatus);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pParsedAddress1</td>
<td>[Input] A pointer to a structure that contains the first part of the decomposed input address.</td>
</tr>
</tbody>
</table>
Parameter | Description
--- | ---
pParsedAddress2 | [Optional Input] A pointer to a structure that contains the second part of the decomposed input address, usually used to contain apartment numbers or post office box numbers. pParsedAddress2 may be NULL.
pLastLine | [Input] A pointer to a structure that contains the decomposed last line components.
pMatchStatus | [Output] A pointer to a variable that receives the match status, which is an enumeration type (see Structures and Defines section on page 327, for specific values).

## Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>Memory allocation of internal structure failed.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory over-write detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>6</td>
<td>Error reading from finance record from database file.</td>
</tr>
<tr>
<td>GEO_ENG_FILE_WRITE_ERR</td>
<td>7</td>
<td>Couldn’t close previous .adx</td>
</tr>
<tr>
<td>GEO_ENG_INDEX_OPEN_ERR</td>
<td>8</td>
<td>Couldn’t open next .adx</td>
</tr>
<tr>
<td>GEO_ENG_END_OF_DATA</td>
<td>11</td>
<td>Premature end of data found.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_POSTAL_CODE</td>
<td>12</td>
<td>Postal code is not valid and city/state is not valid.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_INPUT_ADDRESS</td>
<td>14</td>
<td>Input address is invalid or missing fields.</td>
</tr>
<tr>
<td>GEO_ENG_NO_DATA_AVAILABLE</td>
<td>15</td>
<td>Internal database inconsistency.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_INTERSECT_FORMAT</td>
<td>60</td>
<td>Couldn’t find street or intersection in input.</td>
</tr>
<tr>
<td>GEO_ENG_PARSE_ENGINE_ERR</td>
<td>308</td>
<td>Parsing subsystem error.</td>
</tr>
<tr>
<td>GEO_ENG_NO_STREET_ADDRESS</td>
<td>309</td>
<td>Street address was missing.</td>
</tr>
<tr>
<td>GEO_ENG_PARSE_STACK_OVERFLOW</td>
<td>310</td>
<td>Parser stack overflow (probably bad address).</td>
</tr>
</tbody>
</table>
Structures and Defines

/*
 * Defined constants for the character array sizes in the
 * PARSED_ADDRESS structure.
 */
#define MAX_HOUSE_NUMBER_LEN 25
#define MAX_HOUSE_PREFIX_LEN 11
#define MAX_HOUSE_SEPARATOR_LEN 3
#define MAX_HOUSE_COORD_LEN 8
#define MAX_HOUSE_SUFFIX_LEN 11
#define MAX_STREET_PREDIR_LEN 3
#define MAX_STREET_PRETYPE_LEN 11
#define MAX_STREET_NAME_LEN 29
#define MAX_STREET_POSTTYPE_LEN 5
#define MAX_STREET_POSTDIR_LEN 3
#define MAX_POSTBOX_LEN 8
#define MAX_UNIT_TYPE_LEN 5
#define MAX_UNIT_VALUE_LEN 9

typedef struct
{
    char housePrefix[MAX_HOUSE_PREFIX_LEN];
    char houseNumber[MAX_HOUSE_NUMBER_LEN];
    char houseSeparator[MAX_HOUSE_SEPARATOR_LEN];
    char houseCoord[MAX_HOUSE_COORD_LEN];
    char houseSuffix[MAX_HOUSE_SUFFIX_LEN];
    char preDir[MAX_STREET_PREDIR_LEN];
    char preType[MAX_STREET_PRETYPE_LEN];
    char name[MAX_STREET_NAME_LEN];
    char postType[MAX_STREET_POSTTYPE_LEN];
    char postDir[MAX_STREET_POSTDIR_LEN];
    char postalBox[MAX_POSTBOX_LEN];
    /* no longer used in version 3 */
    char unitType[MAX_UNIT_TYPE_LEN];
    char unitValue[MAX_UNIT_VALUE_LEN];
} PARSED_ADDRESS, *pPARSED_ADDRESS;

/*
 * Defined constants for the character array sizes in the
 * LAST_LINE structure.
 */
#define MAX_CITY_LEN 40
#define MAX_STATE_LEN 40
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10
typedef struct {
    char city[MAX_CITY_LEN];
    char state[MAX_STATE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
} LAST_LINE, *pLAST_LINE;

/*
 * MATCH_STATUS declaration, as well as the Defined constants for the values it can be.
 */
typedef short MATCH_STATUS, *pMATCH_STATUS;

#define INTERSECT_BASE 4
#define SINGLE_MATCH 0
#define MULTIPLE_MATCHES 1
#define NO_MATCHES 2
#define NO_CANDIDATES 3
#define SINGLE_INTERSECT_MATCH 0 + INTERSECT_BASE
#define MULTIPLE_INTERSECT_MATCH 1 + INTERSECT_BASE
#define NO_INTERSECT_MATCHES 2 + INTERSECT_BASE
#define NO_INTERSECT_CANDIDATES 3 + INTERSECT_BASE
#define POSSIBLE_INTERSECTION 4 + INTERSECT_BASE

GeoEngMatchHwyExit()

Purpose

This call uses input criteria of highway exit and state, and returns a count of the number of candidates that matched, if any. A subsequent call to GeoEngGetIndexedHwyExitCandidate() retrieves the matched candidates.

Syntax

  short GeoEngMatchHwyExit(GEO_ENG_HANDLE geoEngHandle,
                          char *hwyExitLine,
                          char *state,
                          short *count)
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
</tbody>
</table>
| hwyExitLine | [Input] A pointer to a string of highway exit information in the form: `<highway name> <highway directional> EXIT<exit number><exit number suffix>TO +<exit name>`. Where:  
  highway name is the name of the highway name including type prefix or type suffix (e.g., Highway 8, I-90, or Pennsylvania Turnpike.  
  highway directional is the direction of the highway, full or abbreviated (e.g., E or East). Optional.  
  EXIT keyword is a case-insensitive keyword used to separate the highway information from the exit information. Required.  
  exit number is the actual number of the exit. This is optional if there is an exit name in the input.  
  exit number suffix is any non-numerical exit suffix (e.g., Exit 2E). Optional, not used by MapMarker.  
  TO keyword is a case-insensitive keyword used to separate the exit name and exit number. Optional.  
  exit name is the name of the exit (e.g., State Hwy 5). Optional, MapMarker will match only to exact names when there is no exit number in the input or there is no match to the exit number in the input.  
  Sample input:  
  I-87 EXIT 2E TO State Hwy 5  
  I-87 EXIT State Hwy 5  
  I-87 EXIT 2E  
  I-87 EXIT 2  
  I-87 EXIT 2E State Hwy .                                                                                                                                                                                                 |}
| state       | [Input] A pointer to the state name, either given in full or abbreviated.                                                                                                                                                                                                                                                                 |
| count       | [Output] The number of candidates that matched the input criteria.                                                                                                                                                                                                                                                                          |

### Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>The normal return for the function. It does not necessarily mean that there was a match.</td>
</tr>
<tr>
<td>NO_MATCH</td>
<td>-1</td>
<td>There was no keyword input; or did not find a match.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_INPUT_ADDRESS</td>
<td>14</td>
<td>Input address is invalid or there are missing fields.</td>
</tr>
<tr>
<td>GEO_ENGPARSE_ENGINE_ERR</td>
<td>308</td>
<td>Parsing subsystem error.</td>
</tr>
</tbody>
</table>
GeoEngNAD27ToNAD83()

Purpose
This call uses the NADCON method to transform coordinate values from the North American Datum of 1927 (NAD 27) to the North American Datum of 1983 (NAD 83).

Syntax

```c
short GeoEngNAD27ToNAD83(GEO_ENG_HANDLE geoEngHandle,
    pDOUBLE_POINT pInputPoint,
    pDOUBLE_POINT pOutputPoint);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pInputPoint</td>
<td>[Input] A pointer to a structure which contains the NAD 27 datum coordinate values.</td>
</tr>
<tr>
<td>pOutputPoint</td>
<td>[Output] A pointer to a structure which receives the NAD 83 datum coordinate values.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>

Structures and Defines

```c
/* DOUBLE_POINT structure definition. It is how we return all coordinates. */
typedef struct {
    double x;
    double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
```
GeoEngNAD83ToNAD27()

Purpose
This call uses the NADCON method to transform coordinate values from the North American Datum of 1983 (NAD 83) to the North American Datum of 1927 (NAD 27).

Syntax
short GeoEngNAD83ToNAD27(GEO_ENG_HANDLE geoEngHandle,
                          pDOUBLE_POINT pInputPoint,
                          pDOUBLE_POINT pOutputPoint);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pInputPoint</td>
<td>[Input] A pointer to a structure which contains the NAD 83 datum coordinate values.</td>
</tr>
<tr>
<td>pOutputPoint</td>
<td>[Output] A pointer to a structure which receives the NAD 27 datum coordinate values.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>

Structures and Defines

```c
/* DOUBLE_POINT structure definition. It is how we return all coordinates.*/
typedef struct {
  double x;
  double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
```
GeoEngParseLastLine()

Purpose
This call accepts a last line string and parses it into city, state, zip, and plus4 components. The string passed into this function is preserved (it works with a copy of it).

Syntax
```
short GeoEngParseLastLine(GEO_ENG_HANDLE geoEngHandle,
    char *pInputLastLine,
    pLAST_LINE pLastLine);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pInputLastLine</td>
<td>[Input] A string which contains the last line to be parsed.</td>
</tr>
<tr>
<td>pLastLine</td>
<td>[Output] A pointer to a structure that receives the decomposed last line components.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>

Structures and Defines

```
/*
 * Defined constants for the character array sizes in the
 * LAST_LINE structure.
 */
#define MAX_CITY_LEN 40
#define MAX_STATE_LEN 40
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10

typedef struct
{
    char city[MAX_CITY_LEN];
    char state[MAX_STATE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
} LAST_LINE, *pLAST_LINE;
```
GeoEngSetCfgParms()

Purpose
This call sets run time parameters for the geocoding engine.

Syntax

```c
short GeoEngSetCfgParms (GEO_ENG_HANDLE geoEngHandle,
pGEO_ENG_CFG_PARMS pGeoEngParms);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the</td>
</tr>
<tr>
<td></td>
<td>geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>pGeoEngParms</td>
<td>[Input] A pointer to a structure containing parameters that affect the behav-</td>
</tr>
<tr>
<td></td>
<td>ior of the geocoding engine.</td>
</tr>
</tbody>
</table>

Note: Keep in mind that if useCassMode is set to TRUE, you are only using CASS rules. You are not actually CASS certified.

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>match.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>2</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>3</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>13</td>
<td>Requested database component failed to initialize.</td>
</tr>
</tbody>
</table>
Structures

/*
* BOOLEAN with TRUE and FALSE.
*/
#define BOOLEAN short

/*
* DIST_UNIT declaration and all the values it can have.
*/
typedef short DIST_UNIT, *pDIST_UNIT;
#define DIST_UNIT_FOOT 0
#define DIST_UNIT_DEGREE 1
#define DIST_UNIT_INCH 2
#define DIST_UNIT_LINK 3
#define DIST_UNIT_SURVEY_FOOT 4
#define DIST_UNIT_YARD 5
#define DIST_UNIT_ROD 6
#define DIST_UNIT_CHAIN 7
#define DIST_UNIT_MILE 8
#define DIST_UNIT_NAUTICAL_MILE 9
#define DIST_UNIT_MILLIMETER 10
#define DIST_UNIT_CENTIMETER 11
#define DIST_UNIT_METER 12
#define DIST_UNIT_KILOMETER 13

typedef struct
{
    double linearInset;
    double perpendicularSetback;
    DIST_UNIT distUnit;
    BOOLEAN relaxHouseNumber;
    BOOLEAN relaxPostalCode;
    BOOLEAN relaxName;
    BOOLEAN relaxFinance;
    BOOLEAN relaxFinanceInState;
    BOOLEAN matchIntersections;
    BOOLEAN useCassMode;
    short relaxDistance;
    BOOLEAN exactCityName;
    BOOLEAN preferUserDictionary;
} GEO_ENG_CFG_PARMS, *pGEO_ENG_CFG_PARMS;

GeoEngSetDpvMode()

Purpose

This call sets DPV mode to on or off. The default is off.

Syntax

    short GeoEngSetDpvMode(GEO_ENG_HANDLE geoEngHandle,
                          BOOLEAN useDpvMode)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>useDpvMode</td>
<td>[Input] Set to TRUE to turn on DpvMode; set to FALSE to turn it off. CASS Mode must be on before DpvMode can be activated. If CASS Mode is not on, and you set DpvMode to TRUE, an error is returned.</td>
</tr>
</tbody>
</table>

GeoEngSetInteractive()

Purpose

This call sets the GeoEngine to do interactive matching. GeoEngSetInteractive() enables the GeoEngine to retrieve the maximum number of candidates from the address dictionary using the current geocoding preferences. After the next call to GeoEngMatchAddress() the GeoEngine is automatically reset.

Syntax

```c
short GeoEngSetInteractive(GEO_ENG_HANDLE geoEngHandle);  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>[Input] An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS</td>
<td>0</td>
<td>Normal return for the function. It does not necessarily mean there was a match.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
</tbody>
</table>
This chapter describes the Data Dictionary API (DD API), a programming interface for creating a customized address dictionary within your application. This API supports the creation of user dictionaries from a MapInfo table. This API works in concert with the MapMarker GeoEngine API, discussed in the previous chapter.

**In this chapter:**

- **Required and Optional Fields** ................................................................. 338
- **User Dictionary File Format** ................................................................. 339
- **Data Dictionary Interface** ................................................................. 340
**Required and Optional Fields**

You must specify the field names in the MapInfo table in order for the table to be translated into a user dictionary. Certain fields are required and must be present in the MapInfo table. Other fields are optional. If any of the required fields are missing, a missing field error code is returned. Below are the required and optional fields. These fields represent one object in the MapInfo table.

**MapInfo Object (Line, Polyline or Point)**

<table>
<thead>
<tr>
<th>Required Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Start Address</td>
<td>Start of address range on left side of street</td>
</tr>
<tr>
<td>Right Start Address</td>
<td>Start of address range on right side of street</td>
</tr>
<tr>
<td>Left End Address</td>
<td>End of address range on left side of street</td>
</tr>
<tr>
<td>Right End Address</td>
<td>End of address range on right side of street</td>
</tr>
<tr>
<td>Street Name</td>
<td>Name of street</td>
</tr>
<tr>
<td>State Abbreviation</td>
<td>Two-character state abbreviation</td>
</tr>
<tr>
<td>Left ZIP Code</td>
<td>ZIP Code for left side of street</td>
</tr>
<tr>
<td>Right ZIP Code</td>
<td>ZIP Code for right side of the street</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Odd/Even indicator*</td>
<td>Left side of the street contains only odd or even address ranges (O=odd, E=even, B=blank)</td>
</tr>
<tr>
<td>Right Odd/Even indicator*</td>
<td>Right side of the street contains only odd or even address ranges (O=odd, E=even, B=blank)</td>
</tr>
<tr>
<td>City*</td>
<td>City name</td>
</tr>
<tr>
<td>Left ZIP+4 Code</td>
<td>4-digit ZIP+4 add-on for left side of street</td>
</tr>
<tr>
<td>Right ZIP+4 Code</td>
<td>4-digit ZIP+4 add-on for right side of street</td>
</tr>
<tr>
<td>Left Census Block</td>
<td>Census Block ID for left side of street</td>
</tr>
<tr>
<td>Right Census Block</td>
<td>Census Block ID for right side of street</td>
</tr>
<tr>
<td>Place Name</td>
<td>Place name</td>
</tr>
</tbody>
</table>

* These fields are highly recommended. See User Dictionary Considerations on page 109, for more information.
Odd/Even Indicator Fields

The Left and Right Odd/Even Indicator fields are used to indicate to MapMarker that the sides of the street segment contain only odd or even address ranges. This indicator is not specified, but MapMarker assumes the value to be either O(dd) if only one of the Start Address and End Address is specified and is odd, or E(ven) if only one of the Start Address and End Address is specified and is even. If this indicator is not specified and both Start Address and End Address have values, MapMarker defaults to the indicator to be "Both" (odd and even). When the indicator is Both, the geocoding process matches to the first range it finds for the record. If all your segments listed with odd ranges first, it puts all the geocoded points on the same side of the street. To avoid this, be sure to use the Odd/Even Indicator fields when designing your user dictionary.

User Dictionary File Format

The user dictionaries that are produced by the DD API are in the same format as the MapMarker Address Dictionary (adr/adx files). The main distinction between the two is that the associated files are named differently. When creating a user dictionary, you must specify a base name of eight characters or less. The name cannot be a duplicate of any MapMarker Address Dictionary file names (i.e., ny12). Also, each user dictionary contains its own zipmastr, cityfin, usafin, and relaxed finance index files. These files are used to look up ZIP Codes or city/state names to find out where to look in the user dictionary file when geocoding. Below is a table that compares the MapMarker Address Dictionary file extensions to a custom user dictionary file set.

<table>
<thead>
<tr>
<th>MapMarker Address Dictionary Naming Scheme</th>
<th>User Dictionary Naming Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY12.ADR</td>
<td>STREETS.UDR</td>
</tr>
<tr>
<td>NY12.ADX</td>
<td>STREETS.UDX</td>
</tr>
<tr>
<td>ZIPMASTR.CDB</td>
<td>STREETS.ZMD</td>
</tr>
<tr>
<td>ZIPMASTR.JDX</td>
<td>STREETS.ZMX</td>
</tr>
<tr>
<td>CITYFIN.JDX</td>
<td>STREETS.CFX</td>
</tr>
<tr>
<td>USAFIN.CDB</td>
<td>STREETS.UFD</td>
</tr>
<tr>
<td>FINCTRDI.DX</td>
<td>STREETS.RFX</td>
</tr>
</tbody>
</table>
## Data Dictionary Interface

The DD API is written at a very high level so that not much user intervention is required once the initial API call has been made. All of the code for the API is integrated into the GeoEngine of MapMarker. When using `GeoEngDDCreateUserDictionary()`, `datadict.lib` must be included in your list of libraries.

### GeoEngDDCreateUserDictionary()

**Purpose**

This function creates a user-defined address dictionary from a MapInfo table to be used in geocoding with MapMarker’s GeoEngine.

**Syntax**

```c
long GeoEngDDCreateUserDictionary(GEO_ENG_HANDLE geoEngHandle,
                                 char *input_table_name,
                                 char *output_dictionary_name,
                                 pUD_INFO   pUDInfo
                                 (long (*pStatusFunction)(long)))
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to <code>GeoEngInit()</code>.</td>
</tr>
<tr>
<td>input_table_name</td>
<td>Full path of the MapInfo input table name from which a user dictionary is created.</td>
</tr>
<tr>
<td>output_dictionary_name</td>
<td>Full path to the output dictionary including the user dictionary file extension <code>.udr</code>.</td>
</tr>
<tr>
<td>pUDInfo</td>
<td>InfoPointer to the UD_Info (user dictionary information) structure</td>
</tr>
<tr>
<td>pStatusFunction</td>
<td>Pointer to status function</td>
</tr>
</tbody>
</table>

This API routine creates a user dictionary from a MapInfo input table. The user dictionary is named according to the `output_dictionary_name` argument given to the routine. The GeoEngine handle is used to look up ZIP Codes in the `zipmastr` file and to look up city names in the `cityfin` file to find corresponding finance numbers. The pointer to the user dictionary info structure must be initialized by the user. This structure must be filled with the table field names that are to be used for translation, some of which are required to be present in the MapInfo source table. An error code is returned if field names are not found for these fields in the UD_Info structure. The process flag in `UD_INFO` should be set to `ALL_ROWS` or `SUBSET_ROWS` with `iStartRow` and `iEndRow` set accordingly.
The status function pointer represents a function that can be called to update the status of the API. It gives the end user the opportunity to cancel the dictionary creation process, if they choose. The status function must take a long integer and return a long integer. It must return a flag, which represents whether or not to cancel the operation. By returning 0, the API continues. However, if 1 is returned then the API halts as if cancelled. The input parameter is a number between 1 and 100 which represents the status of the API in percentage. This value can be used to print the status to the screen.

You also have the option of not using the status function. If the status function pointer is NULL or 0, then the API does not attempt to call the status function.

**Structures and Defines**

The user dictionary creation interface needs to know the field names in the MapInfo table and the starting and ending rows for translation. This information is stored in the Data Dictionary data structure. The DD API uses a single large structure to store all of the options, including the names of the columns, the starting and ending row numbers, and a process flag that indicates how much of the rows should be processed. The structure definition is listed below.

```c
#define MAX_FIELD_NAME_LEN 31
#define ALL_ROWS 1
#define SUBSET_ROWS 2

typedef struct ud_info
{
    char LeftStartAddrFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftEndAddrFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightStartAddrFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightEndAddrFieldName[MAX_FIELD_NAME_LEN + 1];
    char StreetFieldName[MAX_FIELD_NAME_LEN + 1];
    char CityFieldName[MAX_FIELD_NAME_LEN + 1];
    char StateFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftZipCodeFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightZipCodeFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftZipAddOnFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightZipAddOnFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftCensusBlockFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightCensusBlockFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftOddEvenFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightOddEvenFieldName[MAX_FIELD_NAME_LEN + 1];
    char PlaceFieldName[MAX_FIELD_NAME_LEN + 1];
    long iProcessFlag;
    long iStartRow;
    long iEndRow;
} UD_INFO, *pUD_INFO;
```
Part III: Appendixes

Appendix Topics:

- MapMarker User Interface Reference ......................................................... 385
- Data Structures ......................................................................................... 343
- MapMarker Preferences ........................................................................... 369
- Understanding Datums .............................................................................. 375
- Creating a Map Catalog By Hand ............................................................... 379
- Frequently Asked Questions ...................................................................... 383
- MapMarker Utility Programs ..................................................................... 387
- MapMarker Program Files ......................................................................... 393
- MapMarker GeoEngine Error Codes ......................................................... 395
- U.S. ZIP Code Ranges ............................................................................... 411
- Developer Information ............................................................................... 405
- Street Suffix Abbreviations ....................................................................... 415
This section describes each of the data structures that the MapMarker Geoengine API uses. Each language that the Geoengine API supports has a set of data structures to go with it. The structures for each language are located in a separate include file. The include files are located in the include folder. The default location would be C:\Program Files\MapInfo\MapMarker\geoeng\include, or the location where you have MapMarker installed. The languages and their corresponding include files are listed below:

- C/C++–geo.h
- MapBasic–migeo32.def
- Visual Basic–migeo.bas

Complete descriptions are provided for the C/C++ structures. Because the purpose of each structure is the same for each of the languages, and the names of the structures and their members are nearly identical, the structures for MapBasic and Visual Basic are not fully described. However, the names of the corresponding MapBasic and Visual Basic structures are listed at the end of each structure description.
ADDRESS structure

Purpose
Structure that contains the fields that make up an address. The ADDRESS structure is used for both input and output. As input, it contains an address to be matched against the Address Dictionary. As output, it contains the parsed form of the address.

Syntax

```c
/*\n* Defined constants for the character array sizes in the ADDRESS
* structure.
*/
#define MAX_BUSINESS_LEN 256
#define MAX_STREET_LEN 256
#define MAX_CITY_LEN 40
#define MAX_STATE_LEN 40
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10
#define MAX_URBANIZATION_LEN 40

typedef struct
{
    char firm[MAX_BUSINESS_LEN];
    char street[MAX_STREET_LEN];
    char street2[MAX_STREET_LEN];
    char city[MAX_CITY_LEN];
    char state[MAX_STATE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
    char urbanization[MAX_URBANIZATION_LEN];
} ADDRESS, *pADDRESS;
```

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>The name of the firm in the address.</td>
</tr>
<tr>
<td>street</td>
<td>The name of the street in the address.</td>
</tr>
<tr>
<td>city</td>
<td>The name of the city in the address.</td>
</tr>
<tr>
<td>state</td>
<td>The name of the state in the address.</td>
</tr>
<tr>
<td>postalCode</td>
<td>Unique identifier for postal mailing zones. In the United States, it's a 5- or 9-digit ZIP Code.</td>
</tr>
<tr>
<td>postalAddOnCode</td>
<td>Additional postal code identifier. In the United States it is a 4-digit addition to the 5-digit ZIP Code.</td>
</tr>
<tr>
<td>urbanization</td>
<td>Denotes an area, sector, or development within a geographic area, particularly Puerto Rican urban areas. It is an important part of the addressing format that describes the location of a given street. This is a required field when geocoding in CASS mode, even if your table does not contain Puerto Rican addresses.</td>
</tr>
</tbody>
</table>
Methods that Use Structure

GeoEngMatchAddress() on page 322

Languages Used

For MapBasic, see addressRec in migeo32.def.

For Visual Basic, see Address in migeo.bas.

BROWSED_PLACE structure

Purpose

Structure that contains the place name returned from a browse place request via the GeoEngBrowsePlace() function.

Syntax

```c
/*
 * Defined constants for character array sizes in a BROWSED_PLACE
 * structure
 */
#define MAX_PLACE_NAME_LEN 35
typedef struct
{
    char name[MAX_PLACE_NAME_LEN];
} BROWSED_PLACE, *pBROWSED_PLACE;
```

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the browsed place returned from the Address Dictionary.</td>
</tr>
</tbody>
</table>

Methods that Use Structure

GeoEngBrowsePlace() on page 282

Languages Used

For MapBasic, see browsedPlace in migeo32.def.

For Visual Basic, see browsedPlace in migeo.bas.
**BROWSED_STREET structure**

**Purpose**

Structure that contains the street name returned from a browse request via the GeoEngBrowseStreet() function.

**Syntax**

```c
/*
 * Defined constants for the character array sizes in all the
 * BROWSED_STREET structure.
 */
#define MAX_STREET_PREDIR_LEN 3
#define MAX_STREET_PRETYPE_LEN 11
#define MAX_STREET_NAME_LEN 29
#define MAX_STREET_POSTTYPE_LEN 5
#define MAX_STREET_POSTDIR_LEN 3

typedef struct
{
    char preDir[MAX_STREET_PREDIR_LEN];
    char preType[MAX_STREET_PRETYPE_LEN];
    char name[MAX_STREET_NAME_LEN];
    char postType[MAX_STREET_POSTTYPE_LEN];
    char postDir[MAX_STREET_POSTDIR_LEN];
} BROWSED_STREET, *pBROWSED_STREET;
```

**Members**

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preDir</td>
<td>Directional that precedes the street name, e.g., “West” as in “West Tower St.”</td>
</tr>
<tr>
<td>preType</td>
<td>Street type that precedes the street name, e.g., “Ave” as in “Avenue B.”</td>
</tr>
<tr>
<td>name</td>
<td>The name of the browsed street returned from the Address Dictionary.</td>
</tr>
<tr>
<td>postType</td>
<td>Street type that follows the street name, e.g., “St.” as in “Main St.”</td>
</tr>
<tr>
<td>postDir</td>
<td>Directional that follows the street name, e.g., “West” as in “Front St. West.”</td>
</tr>
</tbody>
</table>

**Methods that Use Structure**

GeoEngBrowseStreet() on page 284

**Languages Used**

For MapBasic, see browsedStreet in migeo32.def.

For VisualBasic, see browsedStreet in migeo.bas.
CANDIDATE_ADDRESS structure

Purpose
Structure that groups all address information associated with a potential match to a geocode request.

Syntax

```c
/*
* Defined constants for the addressType field in the
* CANDIDATE_ADDRESS structure.
*/
#define ADDRESS_TYPE_STREET 10
#define ADDRESS_TYPE_PLACE 11
#define ADDRESS_TYPE_ZIP 12
#define ADDRESS_TYPE_RURAL 13
#define ADDRESS_TYPE_HIGHWAY 14
#define ADDRESS_TYPE_POBOX 15
#define ADDRESS_TYPE_MILITARY 16
#define ADDRESS_TYPE_INTERSECTION 17
/*
* The following values are OR'd together to produce a
* fieldsMatchedValue value in the CANDIDATE_ADDRESS structure.
*/
#define NON_CLOSE_MATCH 1
#define CLOSE_MATCH (1 << 1)
#define PREDIR_MATCH (1 << 2)
#define NAME_EXACT_MATCH (1 << 3)
#define POSTDIR_MATCH (1 << 4)
#define TYPE_MATCH (1 << 5)
#define POSTAL_CODE_MATCH (1 << 6)
#define HOUSE_NUMBER_MATCH (1 << 7)
#define USER_DICT_MATCH (1 << 8)
/*
* Defined constants for the character array sizes in the ADDRESS
* structure.
*/
#define MAX_BUSINESS_LEN 256
#define MAX_STREET_LEN 256
#define MAX_CITY_LEN 40
#define MAX_STATE_LEN 40
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10
#define MAX_URBANIZATION_LEN 40
```
typedef struct
{
    char firm[MAX_BUSINESS_LEN];
    char street[MAX_STREET_LEN];
    char street2[MAX_STREET_LEN];
    char city[MAX_CITY_LEN];
    char state[MAX_STATE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
    char urbanization[MAX_URBANIZATION_LEN];
} ADDRESS, *pADDRESS;

/*
* Defined constants for the character array sizes in the
* CANDIDATE_ADDRESS structure.
*/
#define MAX_CENSUS_BLOCK_LEN 20
#define MAX_STREET_LEN 256
#define MAX_LAST_LINE_LEN 40
#define MAX_CARRIER_ROUTE_LEN 5
#define DELIVERY_POINT_LEN 3
#define CHECK_DIGIT_LEN 2
#define MAX_LACS_LEN 2
#define PMB_RANGE_LEN 8
#define PMB_LEN 4

typedef struct
{
    short addressType;
    short fieldsMatchedValue;
    short score;
    BOOLEAN bPmbOutaddr2;
    ADDRESS candidateAddress;
    char censusBlockId[MAX_CENSUS_BLOCK_LEN];
    char tabbedAddress[MAX_STREET_LEN];
    char primaryStreet[MAX_STREET_LEN];
    char lastLine[MAX_LAST_LINE_LEN];
    char deliveryPoint[DELIVERY_POINT_LEN];
    char checkDigit[CHECK_DIGIT_LEN];
    char lacs[MAX_LACS_LEN];
    char carrierRoute[MAX_CARRIER_ROUTE_LEN];
    char record_type;
    char pmb_range[PMB_RANGE_LEN];
    char pmb[PMB_LEN];
} CANDIDATE_ADDRESS, *pCANDIDATE_ADDRESS;

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addressType</td>
<td>Output: Numeric value that indicates the address type as defined by a set of constants.</td>
</tr>
<tr>
<td>fieldsMatchedValue</td>
<td>Output: Aggregate value made up of the match values of the parts of the candidate address.</td>
</tr>
<tr>
<td>score</td>
<td>Output: The match score of the candidate, based on spelling.</td>
</tr>
<tr>
<td>Member Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>bPmbOutaddr2</td>
<td>Output: Boolean value that indicates if the Pmb will be put into the addr2 field.</td>
</tr>
<tr>
<td>candidateAddress</td>
<td>Output: The address of the candidate.</td>
</tr>
<tr>
<td>censusBlockId</td>
<td>Output: Unique U.S. Census Bureau identifier that defines the smallest geographic area for census reporting. The 15-digit alphanumeric code is interpreted as follows: SSSCCTTTTTTGBB(B) where: S = State FIPS code (Federal Information Processing Standard) (2 characters) C = County FIPS code (3 characters) T = Census Tract (6 characters) G = Census Block Group (1 character) B = Census Block (typically 2 characters. The third character means the block is split from another block). The completeness of the returned CensusBlockId is indicated by the result code. An SS5 result indicates that the full 15-digit code to represent the census block was returned; the S2, S3 and Z3 result codes return the block group; S1 returns the census tract.</td>
</tr>
<tr>
<td>tabbedAddress</td>
<td>Output: Street name portion of the address separated by tabs. The street address can contain up to eight parts: house number, prefix direction, prefix type, street name, postfix type, postfix direction, unit type, and unit value. For an intersection, the tabbed address is made up of the two streets joined by the '&amp;' character and surrounded by tabs. See below for more detailed information on the tabbedAddress field.</td>
</tr>
<tr>
<td>primaryStreet</td>
<td>Primary name for a street that has an alias (another name). This is most common along roads that are parts of highways. For example, Central Ave has an alias of State Rt 5. In that case Central Ave is the primary name. You will only get back a value for primaryStreet if you use the alias name.</td>
</tr>
<tr>
<td>lastLine</td>
<td>An input parameter that contains city, state and postal information in the same field.</td>
</tr>
<tr>
<td>deliveryPoint</td>
<td>Output: A two-digit code that, together with Check Digit, is used to form the delivery point barcode on mail pieces. This output component is required when geocoding in CASS mode.</td>
</tr>
<tr>
<td>checkDigit</td>
<td>Output: Part of the delivery point barcode on U.S. mail pieces, along with the Delivery Point Code. This output address component is required when geocoding in CASS mode.</td>
</tr>
<tr>
<td>lacs</td>
<td>Output: A single-character code that indicates that a record has been converted from a rural route address to a city-style address by the USPS through the Locatable Address Conversion Service (LACS). These addresses are converted so that emergency vehicles (e.g., ambulances, police cars, etc.) can find these locations more easily. The field has a value of L or it is blank. This output address component is required if the table is being geocoded for CASS certification.</td>
</tr>
</tbody>
</table>
The tabbedAddress field of the CANDIDATE_ADDRESS structure is an ASCII tab-delimited field. It has two formats. The primary format represents a componentized breakdown of the geocoded address for a candidate. It can consist of the following eight tab-delimited fields:

House Number<tab>Prefix Directional<tab>Prefix Type<tab>Street Name<tab>Postfix Type<tab>Postfix Directional<tab>Unit Type<tab>Unit Value<tab>

Each field contains the following information:

<table>
<thead>
<tr>
<th>Address Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>House Number&lt;tab&gt;</td>
<td>The house number, including any prefix or suffix, e.g., “10A”, as in “10A Main St.”</td>
</tr>
<tr>
<td>Prefix Directional&lt;tab&gt;</td>
<td>Directional that precedes the name, e.g., “West” as in “West Tower St.”</td>
</tr>
<tr>
<td>Prefix Type&lt;tab&gt;</td>
<td>Street type that precedes the name, e.g., “Ave” as in “Avenue B.”</td>
</tr>
<tr>
<td>Street Name&lt;tab&gt;</td>
<td>The core name of the street, e.g., “B” as in “Avenue B.”</td>
</tr>
<tr>
<td>Postfix Type</td>
<td>Street type that follows the name, e.g., “St” as in “Main St.”</td>
</tr>
<tr>
<td>Postfix Directional</td>
<td>Directional that follows the name, e.g., “West,” as in “Front St. West.”</td>
</tr>
<tr>
<td>Unit Type&lt;tab&gt;</td>
<td>The type of unit, e.g., apt., suite, etc.</td>
</tr>
<tr>
<td>Unit Value&lt;tab&gt;</td>
<td>The unit number.</td>
</tr>
</tbody>
</table>

If a field is empty, there will be no value before the tab. The full number of tabs should be present when using this format, even if there are empty fields.

The second format was developed to handle original names of streets. It consists of:

House Number<tab>Full Street Name<tab>

The original name is created during data processing. If the processing changes the street name in some way, it saves the original form as the “original name” and sets a flag. For every internal candidate structure that has the original flag set, the original name of the street is returned instead of the componentized name fields in the structure.
Methods that Use Structure

GeoEngFindPostalCentroid() on page 291
GeoEngGetIndexedCandidate() on page 303

Languages Used
For MapBasic, see candidateAddress in migeo32.def.
For VisualBasic, see CandidateAddress in migeo.bas.

CNTY_NAME_FIPS_REC structure

Purpose
Structure that contains a 5-digit Federal Information Processing Standard (FIPS) code representing the state and county and the county name that matches.

Syntax

typedef struct
{
    Long numCountyFips;
    CNTY_NAME_FIPS_REC countyFipsList[MAX_NUM_COUNTIES];
} COUNTY_FIPS_LIST, *pCOUNTY_FIPS_LIST;

typedef struct
{
    char fipsCode[MAX_FIPS_LEN + 1]
    char cntyName[MAX_CNTY_NAME_LEN + 1];
} CNTY_NAME_FIPS_REC, *pCNTY_NAME_FIPS_REC;

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fipsCode</td>
<td>The five-digit Federal Information Processing Standard (FIPS) code for the county and state.</td>
</tr>
<tr>
<td>cntyName</td>
<td>The name of the county.</td>
</tr>
</tbody>
</table>

Methods that Use Structure
GeoEngGetZIPCountyFIPSList() on page 313

Languages Used
For MapBasic, see CountyFipsList in migeo32.def.
For Visual Basic, see countyFipsList in migeo.bas.
DOUBLE_POINT structure

Purpose

Structure that contains the longitude/latitude coordinates of the candidate point.

Syntax

```c
/*
 * Precision levels for Coordinates returned by the engine.
 */
#define NO_CENTROID 0
#define ZIPCODE_CENTROID 1
#define ZIPPLUS2_CENTROID 2
#define ZIPPLUS4_CENTROID 3
#define SHAPE_PATH_CENTER_COORDS 10
#define STREET_ADDRESS_COORDS 20
#define STREET_INTERSECT_COORDS 30
#define POINT_ZIP 40
/
* DOUBLE_POINT structure definition. It is how we return all
 * coordinates.
 */
typedef struct {
    double x;
    double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
```

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>The latitude coordinate of the candidate point.</td>
</tr>
<tr>
<td>y</td>
<td>The longitude coordinate of the candidate point.</td>
</tr>
</tbody>
</table>

Methods that Use Structure

- `GeoEngFindPostalCentroid()` on page 291
- `GeoEngGetCandidateCoords()` on page 297
- `GeoEngGetIndexedCoords()` on page 306
- `GeoEngNAD27ToNAD83()` on page 330
- `GeoEngNAD83ToNAD27()` on page 331

Languages Used

For MapBasic, see pointRec in migeo32.def.
For Visual Basic, see PointRec in migeo.bas.
DPV Info structure

Structure that contains the DPV information for the candidate.

Syntax

```c
#define MAX_FOOTNOTE_LEN 2
typedef struct {
    char dpvCode
    char cmra
    char falsePositive
    char footnote1 [MAX_FOOTNOTE_LEN + 1]
    char footnote2 [MAX_FOOTNOTE_LEN + 1]
    char footnote3 [MAX_FOOTNOTE_LEN + 1]
    char footnote4 [MAX_FOOTNOTE_LEN + 1]
    char footnote5 [MAX_FOOTNOTE_LEN + 1]
    char footnote6 [MAX_FOOTNOTE_LEN + 1]
} DPV_INFO, *pDPV_INFO;
```

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpvCode</td>
<td>The DPV return code for the candidate. The return code can be one of the following: N–This address does not exist. Y–This address exists. S–The street address exists, but the unit does not. D–Address is incomplete (highrise with no unit, or rural route with no box number).</td>
</tr>
<tr>
<td>cmra</td>
<td>Indicates if the candidate belongs to a Commercial Mail Receiving Agency (CMRA)</td>
</tr>
<tr>
<td>falsePositive</td>
<td>If true, DPV has detected a condition in which the candidate appears to be artificially generated and not a legitimately obtained address. DPV will shut down immediately in this instance.</td>
</tr>
<tr>
<td>footnote1</td>
<td>Field that may be populated with a standard USPS footnote code. Each of the codes is described below.</td>
</tr>
<tr>
<td>footnote2</td>
<td>Field that may be populated with a standard USPS footnote code. Each of the codes is described below.</td>
</tr>
<tr>
<td>footnote3</td>
<td>Field that may be populated with a standard USPS footnote code. Each of the codes is described below.</td>
</tr>
<tr>
<td>footnote4</td>
<td>Field that may be populated with a standard USPS footnote code. Each of the codes is described below.</td>
</tr>
<tr>
<td>footnote5</td>
<td>Field that may be populated with a standard USPS footnote code. Each of the codes is described below.</td>
</tr>
<tr>
<td>footnote6</td>
<td>Field that may be populated with a standard USPS footnote code. Each of the codes is described below.</td>
</tr>
</tbody>
</table>
Footnote Codes
AA–Input address matched to the ZIP+4 file.
A1–Input address not matched to the ZIP+4 file.
BB–Input address matched to DPV (all components).
CC–Input address primary number matched to DPV but secondary number not matched (present but invalid).
N1–Input address primary number matched to DPV but highrise address missing secondary number.
M1–Input address primary number missing.
M3–Input address primary number invalid.
P1–Input address missing PO, RR, or HC box number.
RR–Input address matched to CMRA.
R1–Input address matched to CMRA but secondary number not present.

Methods that Use Structure
GeoEngGetCandidateDpvInfo() on page 300

Languages Used

GEO_ENG_CFG_PARMS structure

Purpose
Structure that defines the engine configuration settings, such as relaxHouseNumber, useCassMode, or matchIntersections.

Syntax
/*
 * BOOLEAN with TRUE and FALSE.
 */
#define BOOLEAN short
/*
 * DIST_UNIT declaration and all the values it can have.
 */
typedef short DIST_UNIT, *pDIST_UNIT;
#define DIST_UNIT_FOOT 0
#define DIST_UNIT_DEGREE 1
#define DIST_UNIT_INCH 2
#define DIST_UNIT_LINK 3
#define DIST_UNIT_SURVEY_FOOT 4
#define DIST_UNIT_YARD 5
#define DIST_UNIT_ROD 6
#define DIST_UNIT_CHAIN 7
#define DIST_UNIT_MILE 8
#define DIST_UNIT_NAUTICAL_MILE 9
#define DIST_UNIT_MILLIMETER 10
#define DIST_UNIT_CENTIMETER 11
#define DIST_UNIT_METER 12
#define DIST_UNIT_KILOMETER 13
typedef struct
{
    double linearInset;
    double perpendicularSetback;
    DIST_UNIT distUnit;
    BOOLEAN relaxHouseNumber;
    BOOLEAN relaxPostalCode;
    BOOLEAN relaxName;
    BOOLEAN relaxFinance;
    BOOLEAN relaxFinanceInState;
    BOOLEAN matchIntersections;
    BOOLEAN useCassMode;
    short relaxDistance;
    BOOLEAN exactCityName;
    BOOLEAN preferUserDictionary;
} GEO_ENG_CFG_PARMS, *pGEO_ENG_CFG_PARMS;

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linearInset</td>
<td>Distance that the point lies from a street corner. This prevents the first</td>
</tr>
<tr>
<td></td>
<td>house on two cross streets from displaying on top of one another.</td>
</tr>
<tr>
<td>perpendicularSetback</td>
<td>Distance from the center line of the street in a perpendicular direction.</td>
</tr>
<tr>
<td></td>
<td>This allows points to display on the left and right sides of the street.</td>
</tr>
<tr>
<td>distUnit</td>
<td>Output: Unit of measurement that describes the offset distance of the</td>
</tr>
<tr>
<td></td>
<td>point from the street.</td>
</tr>
<tr>
<td>relaxHouseNumber</td>
<td>Input: When set to True, the engine will not require an exact match on</td>
</tr>
<tr>
<td></td>
<td>house number.</td>
</tr>
<tr>
<td>relaxPostalCode</td>
<td>Input: When set to True, the engine will not require an exact match on</td>
</tr>
<tr>
<td></td>
<td>postal code.</td>
</tr>
<tr>
<td>relaxName</td>
<td>Input: When set to True, the engine will not require an exact match on</td>
</tr>
<tr>
<td></td>
<td>street name.</td>
</tr>
<tr>
<td>relaxFinance</td>
<td>Input: A flag for the engine to search an expanded area up to the distance</td>
</tr>
<tr>
<td></td>
<td>of the relaxDistance setting. This area equals all the finance areas to</td>
</tr>
<tr>
<td></td>
<td>search based on the input city and ZIP Code plus a boxed area around those</td>
</tr>
<tr>
<td></td>
<td>finance areas of a size equal to the distance set in relaxDistance.</td>
</tr>
<tr>
<td>relaxFinanceInState</td>
<td>Input: A flag for the engine to search the expanded area set to the input</td>
</tr>
<tr>
<td></td>
<td>state.</td>
</tr>
<tr>
<td>matchIntersections</td>
<td>An engine setting for geocoding street intersections.</td>
</tr>
<tr>
<td>useCassMode</td>
<td>Input: When set to True, the engine geocodes using strict CASS matching</td>
</tr>
<tr>
<td></td>
<td>requirements, including exact match on house number, street name, and</td>
</tr>
<tr>
<td></td>
<td>postal code. When this flag is set, relaxHouse, relaxStreet, relaxPostalCode, relaxFinance, and relaxDistance flags are ignored. Default multiple match handling (accept first on multiple match, or pick street or postal code over the other) is also ignored. CASS (Coding Accuracy Support System) is a certification program of the U.S. Postal Service for bulk mailing discounts.</td>
</tr>
</tbody>
</table>
HOUSE_RANGE structure

Purpose
Structure that contains the information on a set of house number ranges found along a segment. A segment usually has multiple house number ranges because odd and even ranges are separate.

Syntax

```c
/*
 * Defined constants for the character array sizes in all the
 * HOUSE_RANGE structure.
 */
#define MAX_HOUSE_NUMBER_LEN 25
#define MAX_UNIT_TYPE_LEN 5
#define MAX_UNIT_VALUE_LEN 9
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10
#define MAX_PLACE_NAME_LEN 35

typedef struct
{
    HOUSE_PARITY houseParity;
    char fromHouse[MAX_HOUSE_NUMBER_LEN];
    char toHouse[MAX_HOUSE_NUMBER_LEN];
    char unitType[MAX_UNIT_TYPE_LEN];
    char fromUnit[MAX_UNIT_VALUE_LEN];
    char toUnit[MAX_UNIT_VALUE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
    char placeName[MAX_PLACE_NAME_LEN];
    char lacs;
    char carrRoute[MAX_CARRIER_ROUTE_LEN];
} HOUSE_RANGE, *pHOUSE_RANGE;
```

Languages Used
For MapBasic, see engineParmsRec in migeo32.def.

For Visual Basic, see EngineParmsRec in migeo.bas.
Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromHouse</td>
<td>The lowest house number found on that segment.</td>
</tr>
<tr>
<td>toHouse</td>
<td>The highest house number found on the segment.</td>
</tr>
<tr>
<td>fromUnit</td>
<td>The lowest unit value for an address range.</td>
</tr>
<tr>
<td>toUnit</td>
<td>The highest unit value for a segment.</td>
</tr>
<tr>
<td>placeName</td>
<td>Names of places, such as office buildings, locally known locations, or businesses.</td>
</tr>
<tr>
<td>lacs</td>
<td>Output: A single-character code that indicates that a record has been converted from a rural route address to a city-style address by the USPS through the Locatable Address Conversion Service (LACS). These addresses are converted so that emergency vehicles (e.g., ambulances, police cars, etc.) can find these locations more easily. The field has a value of L or it is blank. This output address component is required if the table is being geocoded for CASS certification.</td>
</tr>
<tr>
<td>carrierRoute, carrRoute</td>
<td>Output: USPS address element that describes a given mail delivery or collection route within a 5-digit ZIP Code. There are five possible Carrier Route codes: Bnnn = P.O. Box (where n is a number); Hnnn = Highway Contract; Rnnn = Rural Route; Cnnn = City Delivery; Gnnn = General Delivery.</td>
</tr>
</tbody>
</table>

Methods that Use Structure

GeoEngGetNextHouseRange() on page 310

Languages Used

For MapBasic, see houseRange in migeo32.def.

For Visual Basic, see HouseRange in migeo.bas.
HWYEXIT_RECORD structure

Purpose
Structure that contains specific highway exit information: highway name, exit number, and/or exit name, ZIP Code, city, and coordinates of the exit location of the matched candidate.

Syntax

```c
#define EXIT_ID_LEN 10
#define HWY_PRETYPE_LEN 20
#define HWY_NAME_LEN 40
#define HWY_POSTTYPE_LEN 20
#define HWYDIR_LEN 2
#define EXIT_NUM_LEN 10
#define CITY_LEN 40
#define STATE_LEN 2
#define ZIPCODE_LEN 5

typedef struct
{
    char exitID[EXIT_ID_LEN+1];
    char hwyTypePre[HWY_PRETYPE_LEN+1];
    char hwyName[HWY_NAME_LEN+1];
    char hwyTypeSuf[HWY_POSTTYPE_LEN+1];
    char exitNum[EXIT_NUM_LEN+1];
    char exitName[EXIT_NAME_LEN+1];
    char city[CITY_LEN+1];
    char state[STATE_LEN+1];
    char zipcode[ZIPCODE_LEN+1];
    double X;
    double Y;
} HWYEXIT_RECORD, *pHWYEXIT_RECORD;
```
Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitID</td>
<td>The record ID of the highway exit in the highway exit table.</td>
</tr>
<tr>
<td>hwyTypePre</td>
<td>Highway type that precedes the highway name.</td>
</tr>
<tr>
<td>hwyName</td>
<td>The name of the highway.</td>
</tr>
<tr>
<td>hwyTypeSuf</td>
<td>Highway type that follows the highway name.</td>
</tr>
<tr>
<td>extNum</td>
<td>The number of the exit.</td>
</tr>
<tr>
<td>exitName</td>
<td>The name of the exit.</td>
</tr>
<tr>
<td>city</td>
<td>The name of the city of the exit location</td>
</tr>
<tr>
<td>state</td>
<td>The name of the state of the exit location.</td>
</tr>
<tr>
<td>zipcode</td>
<td>The ZIP Code of the exit location.</td>
</tr>
<tr>
<td>X</td>
<td>The latitude coordinate of the exit location.</td>
</tr>
<tr>
<td>Y</td>
<td>The longitude coordinate of the exit location.</td>
</tr>
</tbody>
</table>

Methods that Use Structure

GeoEngGetIndexedHwyExit() on page 308

Languages Used

For MapBasic, see HwyExitRecord in migeo32.def.

For VisualBasic, see HwyExitRecord in migeo.bas.
LAST_LINE structure

Purpose
Structure that is used when the input field contains the city, state, and postal code information in one field instead of separate fields.

Syntax
```c
/*
* Defined constants for the character array sizes in the
* LAST_LINE structure.
*/
#define MAX_CITY_LEN 40
#define MAX_STATE_LEN 40
#define MAX_POSTAL_CODE_LEN 10
#define MAX_POSTAL_ADDON_CODE_LEN 10

typedef struct
{
    char city[MAX_CITY_LEN];
    char state[MAX_STATE_LEN];
    char postalCode[MAX_POSTAL_CODE_LEN];
    char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
} LAST_LINE, *pLAST_LINE;
```

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>city</td>
<td>The name of the city.</td>
</tr>
<tr>
<td>state</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>postalCode</td>
<td>Unique identifier for postal mailing zones. In the United States, it's a 5- or 9-digit ZIP Code.</td>
</tr>
<tr>
<td>postalAddOnCode</td>
<td>Additional postal code identifier. In the United States it is a 4-digit addition to the 5-digit ZIP Code.</td>
</tr>
</tbody>
</table>

Methods that Use Structure

- GeoEngMatchFormattedAddress() on page 325
- GeoEngParseLastLine() on page 332

Languages Used
For MapBasic, see LastLine in migeo32.def.
For Visual Basic, see lastLine in migeo.bas.
MEM_AIRPORT_REC structure

Purpose
Returns information for all airports in a given state using a valid two-letter state abbreviation as input.

Syntax
```c
#define LOC_ID_KEY_LEN 4
#define LOC_ID_LEN 4
#define AIRPORT_NAME_LEN 55
#define CFCC_LEN 3
#define USE_LEN 2
#define OWNERNAME_LEN 29
#define CONGES_LEV_LEN 1
#define SEVR_LEV_LEN 2
#define HUB_SIZE_LEN 1
#define SEVR_LEV_LEN 2
#define OWNERNAME_LEN 29
#define CONGES_LEV_LEN 1
#define SEVR_LEV_LEN 2
#define STATE_LEN 2
#define STATE_KEY_LEN 2
#define FIPS_CNTY_LEN 5

typedef struct
{
    char locId[LOC_ID_LEN + 1];
    char airportName[AIRPORT_NAME_LEN + 1];
    char CFCC[CFCC_LEN + 1];
    char use[USE_LEN + 1];
    char ownerName[OWNERNAME_LEN + 1];
    char congesLev[CONGES_LEV_LEN + 1];
    char servLev[SEVR_LEV_LEN + 1];
    char hubSize[HUB_SIZE_LEN + 1];
    char towerType[TOWER_TYPE_LEN + 1];
    char state[STATE_LEN + 1];
    char FIPSCnty[FIPS_CNTY_LEN + 1];
    double X;
    double Y;
    int id;
    int elevation;
    int lrgCertEnp;
    int commEnp;
    int airTaxiEnp;
    int foreignEnp;
    int inTranEnp;
    int iRecNo;
} MEM_AIRPORT_REC, *pMEM_AIRPORT_REC;
```

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locId</td>
<td>Three or four-character FAA-assigned identifier for the airport, for example, LAX is the locID for Los Angeles International Airport.</td>
</tr>
<tr>
<td>airportName</td>
<td>The name of an airport in a specified state.</td>
</tr>
<tr>
<td>Member Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>CFCC</td>
<td>Census Feature Class Code: A three-character code that identifies the most notable characteristics of a feature.</td>
</tr>
<tr>
<td>use</td>
<td>Indicates whether the airport is public or private. PU—public PR—private</td>
</tr>
<tr>
<td>ownerName</td>
<td>Government or corporate entity that owns or controls the airport.</td>
</tr>
<tr>
<td>congesLev</td>
<td>Indicates the level of congestion at a given airport. Possible returns include: S – Severe M – Moderate U – Uncongested</td>
</tr>
<tr>
<td>servLev</td>
<td>Indicates the level of service at a given airport. Possible returns include: PR – primary commercial CM – commercial CR – commercial reliever GA – general aviation RL – general aviation reliever</td>
</tr>
<tr>
<td>hubSize</td>
<td>Indicates the airport hub size based on the percentage of national enplanements. Possible returns include: L – large M – medium S – small N – non-hub G – general aviation</td>
</tr>
<tr>
<td>state</td>
<td>The two-letter state abbreviation.</td>
</tr>
<tr>
<td>FIPSCnty</td>
<td>The Federal Information Processing Standard (FIPS) code for a given airport.</td>
</tr>
</tbody>
</table>
### Methods that Use Structure

- **GeoEngBrowseAirportByState()** on page 280
- **GeoEngGetAirport()** on page 294

### Languages Used

- For MapBasic, see AirportRecord in migeo32.def.
- For Visual Basic, see AirportRecord in migeo.bas.

### PARSED_ADDRESS

#### Purpose

Structure that contains a street name divided into its parts. A street name can have up to eight parts: house number, prefix direction, prefix type, street name, postfix type, postfix direction, unit type, and unit value. GeoEngMatchAddress() returns the passed address, and GeoEngMatchFormattedAddress() takes it as an argument. In output, the different parts of the address are parsed by the engine.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>The latitude coordinate of the airport location.</td>
</tr>
<tr>
<td>Y</td>
<td>The longitude coordinate of the airport location.</td>
</tr>
<tr>
<td>id</td>
<td>The ID number assigned to each airport</td>
</tr>
<tr>
<td>elevation</td>
<td>The elevation of the airport measured in feet above sea level.</td>
</tr>
<tr>
<td>IrgCertEnp</td>
<td>1992 large certified air carrier enplanings.</td>
</tr>
<tr>
<td>CommEnp</td>
<td>1992 commuter enplanings.</td>
</tr>
<tr>
<td>AirTaxiEnp</td>
<td>1992 air taxi enplanings.</td>
</tr>
<tr>
<td>ForeignEnp</td>
<td>1992 foreign enplanings.</td>
</tr>
<tr>
<td>InTranEnp</td>
<td>1992 in-transit enplanings.</td>
</tr>
<tr>
<td>iRecNo</td>
<td>The record number.</td>
</tr>
</tbody>
</table>
Syntax

/*
 * Defined constants for the character array sizes in the
 * PARSED_ADDRESS structure.
 */
#define MAX_HOUSE_NUMBER_LEN 25
#define MAX_HOUSE_PREFIX_LEN 11
#define MAX_HOUSE_SEPARATOR_LEN 3
#define MAX_HOUSE_COORD_LEN 8
#define MAX_HOUSE_SUFFIX_LEN 11
#define MAX_STREET_PREDIR_LEN 3
#define MAX_STREET_PRETYPE_LEN 11
#define MAX_STREET_NAME_LEN 29
#define MAX_STREET_POSTTYPE_LEN 5
#define MAX_STREET_POSTDIR_LEN 3
#define MAX_POSTBOX_LEN 8
#define MAX_UNIT_TYPE_LEN 5
#define MAX_UNIT_VALUE_LEN 9

typedef struct
{
    char housePrefix[MAX_HOUSE_PREFIX_LEN];
    char houseNumber[MAX_HOUSE_NUMBER_LEN];
    char houseSeparator[MAX_HOUSE_SEPARATOR_LEN];
    char houseCoord[MAX_HOUSE_COORD_LEN];
    char houseSuffix[MAX_HOUSE_SUFFIX_LEN];
    char preDir[MAX_STREET_PREDIR_LEN];
    char preType[MAX_STREET_PRETYPE_LEN];
    char name[MAX_STREET_NAME_LEN];
    char postType[MAX_STREET_POSTTYPE_LEN];
    char postDir[MAX_STREET_POSTDIR_LEN];
    char postalBox[MAX_POSTBOX_LEN];
    /*no longer used in version 3*/
    char unitType[MAX_UNIT_TYPE_LEN];
    char unitValue[MAX_UNIT_VALUE_LEN];
} PARSED_ADDRESS, *pPARSED_ADDRESS;

Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>housePrefix</td>
<td>Character that precedes the house number in a street address.</td>
</tr>
<tr>
<td>houseNumber</td>
<td>Number of the street address location.</td>
</tr>
<tr>
<td>houseSeparator</td>
<td>Character that serves to separate the house prefix from the house number.</td>
</tr>
<tr>
<td>houseCoord</td>
<td>No longer used.</td>
</tr>
<tr>
<td>houseSuffix</td>
<td>Trailing alphabetic and numeric portion of the house number.</td>
</tr>
<tr>
<td>preDir</td>
<td>Describes the directional that precedes a street name, such as N Ave of the Americas, where N is the preDir.</td>
</tr>
<tr>
<td>preType</td>
<td>Describes the street type that precedes a street name, such as N Ave of the Americas, where Ave is the preType.</td>
</tr>
</tbody>
</table>
**UD_INFO**

**Purpose**

Structure that contains the field information used in creating a user dictionary, plus row processing information that indicates which rows in the table are going to be used to create the user dictionary.
## Syntax

```c
#define MAX_FIELD_NAME_LEN 31
#define ALL_ROWS 1
#define SUBSET_ROWS 2

typedef struct ud_info
{
    char LeftStartAddrFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftEndAddrFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightStartAddrFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightEndAddrFieldName[MAX_FIELD_NAME_LEN + 1];
    char StreetFieldName[MAX_FIELD_NAME_LEN + 1];
    char CityFieldName[MAX_FIELD_NAME_LEN + 1];
    char StateFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftZipCodeFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightZipCodeFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftZipAddOnFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightZipAddOnFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftCensusBlockFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightCensusBlockFieldName[MAX_FIELD_NAME_LEN + 1];
    char LeftOddEvenFieldName[MAX_FIELD_NAME_LEN + 1];
    char RightOddEvenFieldName[MAX_FIELD_NAME_LEN + 1];
    char PlaceFieldName[MAX_FIELD_NAME_LEN + 1];
    long iProcessFlag;
    long iStartRow;
    long iEndRow;
} UD_INFO, *pUD_INFO;
```

## Members

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftStartAddrFieldName</td>
<td>Start of address range on left side of street.</td>
</tr>
<tr>
<td>LeftEndAddrFieldName</td>
<td>End of address range on left side of street.</td>
</tr>
<tr>
<td>RightStartAddrFieldName</td>
<td>Start of address range on right side of street.</td>
</tr>
<tr>
<td>RightEndAddrFieldName</td>
<td>End of address range on right side of street.</td>
</tr>
<tr>
<td>StreetFieldName</td>
<td>Name of street.</td>
</tr>
<tr>
<td>CityFieldName</td>
<td>Name of city.</td>
</tr>
<tr>
<td>StateFieldName</td>
<td>Two-character state abbreviation</td>
</tr>
<tr>
<td>LeftZipCodeFieldName</td>
<td>ZIP Code for left side of street.</td>
</tr>
<tr>
<td>RightZipCodeFieldName</td>
<td>ZIP Code for right side of the street.</td>
</tr>
<tr>
<td>LeftZipAddOnFieldName</td>
<td>Four-digit ZIP+4 add-on for left side of street.</td>
</tr>
<tr>
<td>RightZipAddOnFieldName</td>
<td>Four-digit ZIP+4 add-on for right side of street.</td>
</tr>
<tr>
<td>PlaceFieldName</td>
<td>Place name.</td>
</tr>
</tbody>
</table>
Methods that Use Structure

GeoEngDDCreateUserDictionary() on page 340

Languages Used

For MapBasic, see UserDictInfo in migeo32.def.

For VisualBasic, see UserDictInfo in migeo.bas.

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iProcessFlag</td>
<td>Indicates how many of the rows should be processed. Set this flag to ALL_ROWS to process all the rows, or set it to SUBSET_ROWS. When you use SUBSET_ROWS, you must indicate start and end rows using the iStartRow and iEndRow flags.</td>
</tr>
<tr>
<td>iStartRow</td>
<td>The starting row in your subset.</td>
</tr>
<tr>
<td>iEndRow</td>
<td>The ending row in your subset.</td>
</tr>
</tbody>
</table>
MapMarker Preferences

This chapter discusses the setup and geocoding preferences that allow you to customize MapMarker.

In this chapter:

- Setting System Preferences ................................................................. 370
- Startup Preferences ................................................................. 370
- Dictionary Preferences ................................................................. 371
- Offset Preferences ................................................................. 372
- Datum Preferences ................................................................. 372
- Maps Preferences ................................................................. 373
Setting System Preferences

The System Preferences dialog box is displayed when you click **SYSTEM PREFERENCES** on the **OPTIONS** menu, click ![System Preferences](image) or click the secondary mouse button to display a shortcut menu and then click **SYSTEM PREFERENCES**. The System Preferences dialog box contains five tabs: Startup, Dictionary, Offset, Datum, and Maps.

Any changes you make in these tabs are written to the Windows Registry at

```
HKEY_LOCAL_MACHINE\SOFTWARE\MAPINFO\MAPMARKER\<version number>\SYSTEM
```

These changes are also saved for future geocoding sessions.

**Note:** In references to the MapMarker registry keys, `<version number>` indicates the major-release version of MapMarker, e.g., 10.0, even if you are running one of the point releases of that version, e.g., 10.2, 10.3.

Startup Preferences

The **STARTUP** tab identifies the settings MapMarker uses when you start the program. You can select any and all check boxes in the tab. For the file open check boxes, if you select either SHOW MAPINFO/DBASE FILES or SHOW FOR ODBC TABLES, the Open dialog box is displayed automatically when you start MapMarker. If you have both check boxes selected, MapMarker prompts you with another dialog box at startup to choose which type of table you want to open.

You can also control which of the setup dialog boxes, if any, display on startup. Choose from:

- Show Select Column Dialog Box On Open (selected by default)
- Select Attribute Files After Selecting Columns
- Show Geocode Dialog Box After Selecting Columns (selected by default)
- Show Add Georeresult, Longitude/Latitude Columns Dialog (selected by default)

The settings are saved to the table as metadata and used as new defaults when you open the table again.
Dictionary Preferences

The Address Dictionary group identifies the path of all installed address dictionaries. You can purchase address dictionaries from MapInfo on a state-by-state basis, or for the entire United States. The path(s) reflects all address dictionaries you identified during installation, whether the data was copied to a fixed drive or left on CD-ROM. You can add additional paths to the list at any time by separating each path with a semicolon. You do not need to specify the file name.

You can also identify up to five customized dictionaries that contain street addresses that may not be included in the purchased Address Dictionary. To use a customized dictionary, select the check box marked **USE USER DICTIONARY** and type the path or browse to its location. You can geocode
using both the MapMarker Address Dictionary and a customized dictionary. The customized
dictionary takes precedence when both dictionaries encounter a match. Instructions for creating a
user dictionary are provided in Chapter 8: User Dictionaries.

Offset Preferences

These values reflect the distance the geocoded point spots from the corner and center line of the
road. When changing the units of the offset distance, be sure to also type the appropriate offset
distance values, as changing the units does not automatically recalculate the offset values.

Datum Preferences

You can choose whether MapMarker returns matching coordinates with the datum in NAD83 or
NAD27. The NAD83 datum standard reflects a slight recalculation of geographic coordinates from
NAD27. The difference is in how a record matched to the same address from MapMarker can spot
at one location using NAD27 coordinates and at a slightly different location with NAD83
coordinates. Consider how you plan to display your geocoded points. If your street data is in
NAD27, geocode your table using the NAD27 setting. Keep in mind that the standard for TIGER
2000 data is NAD83. MapMarker Streets, MapMarker Plus Streets, StreetInfo 4.0 and later, and all
versions of StreetPro use NAD83 as their datum.
To choose the datum settings, select the item in the Datum dialog box that best meets your needs. To keep your coordinates meaningful, geocode the entire table using the same datum setting. On previously geocoded tables, be sure to know the datum for those geocoded records.

For more on datums, see Appendix C: Understanding Datums.

Maps Preferences

In order to display match candidates in a Map window from the INTERACTIVE or QUICK FIND dialog boxes, MapMarker must know where the street data is located on your system. The Maps tab on the System Preferences dialog box allows you to set the directory path to the street data.
It also enables you to set an initial display radius for the Candidate Visualization Map window. The radius is the distance in miles around the first match candidate that MapMarker finds among multiple matches when you geocode interactively. Set this distance in the Maximum Candidate Map Zoom Distance field. The default value is five miles. You can set the distance from 1–99 miles. Use the zoom in and zoom out tools in the Candidate Visualization Map window to control the view of the map, and choose how much data you want to view in the Map window.

Adjust the amount of data displayed by changing the setting in the Maximum Candidate Map Zoom Distance field. The default value is five miles. The maximum value is 99 miles.

For best results when geocoding with the MapMarker Standard product, use MapMarker Streets or StreetInfo Display as the background street map. These street packages are geographically consistent with the street data in the MapMarker Address Dictionary. MapMarker Streets is included in the MapMarker package.

If you are geocoding against the MapMarker Address Dictionary, be sure to display your geocoded records over an enhanced street product such as MapMarker Plus Streets or StreetPro Enhanced or Display.

For more information on how to map match candidates, see Viewing Match Candidates on a Map on page 57.
Understanding Datums

In this appendix:

- Overview of Datums ................................................................. 376
- NAD83 and the MapMarker Address Dictionary ............................... 376
- How MapMarker Determines Which Datum to Use .......................... 377
- Datums and ODBC Tables ........................................................ 378
Overview of Datums

A datum is a reference point from which geographic coordinates are measured. The latest version of the U.S. Census Bureau's TIGER Line data uses as its standard the North American Datum of 1983, known as NAD83.

NAD83 represents the most accurate and comprehensive geodetic survey for the U.S. to date. It is based on a model that calculates the earth's ellipsoid slightly differently than the old standard NAD27. It provides a consistent measurement across North America at a higher degree of accuracy.

NAD83 and the MapMarker Address Dictionary

MapMarker's Address Dictionary was created with GDT Dynamap/2000 v. 14.1 April 2003 (USPS currentness November 2003) source data that uses NAD83 as its datum reference. This means that the longitude and latitude coordinates that MapMarker assigns to geocoded records use the NAD83 datum and differ slightly from records previously geocoded against NAD27-based Address Dictionaries (such as MapMarker 2.1 or earlier). Coordinate pairs in NAD83 and NAD27 differ on average by +/- 5 meters.

Because of this transition from one datum standard to another, MapMarker allows you to control which datum you want for your geocoded records. Your decision to use NAD83 or NAD27 rests with how you intend to display your geocoded records and whether the point differential affects your analysis.

For example, if you display your records against street data based on NAD27 datum, then you should geocode your table to that datum. If your street display data is in NAD83, geocode to NAD83 so that your points and the street data match. If you have previously geocoded a table with NAD 27, continue to use that datum when you geocode the table again. The important thing to keep in mind is that you geocode using the datum that matches your street data.

The following table lists the datums for MapInfo’s street products:

<table>
<thead>
<tr>
<th>Street Product</th>
<th>Datum</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreetInfo V 2.0 and 3.0</td>
<td>NAD27</td>
</tr>
<tr>
<td>StreetWorks 1.0 and 2.0</td>
<td>NAD27</td>
</tr>
<tr>
<td>MapMarker Streets - all versions</td>
<td>NAD83</td>
</tr>
<tr>
<td>MapMarker Plus Streets - all versions</td>
<td>NAD83</td>
</tr>
<tr>
<td>StreetWorks V 3.0 and later</td>
<td>NAD83</td>
</tr>
</tbody>
</table>
How MapMarker Determines Which Datum to Use

MapMarker looks in the table’s metadata for a datum key and in the .map file (if one exists) and compares them against the setting in the System Preferences>Datum dialog box. If there is a datum mismatch, a “pre-scan” message is displayed, prompting you to resolve the conflict.

For example, if your table has a metadata datum key of NAD27 and the Datum Preferences dialog box is set to Convert all tables to NAD83, the pre-scan message box is displayed. If you answer Yes to convert to NAD83, MapMarker geocodes records to that datum. If you answer No, MapMarker overrides the Datum Preference setting and geocodes your table to the original NAD27 datum.

If MapMarker cannot determine the datum for a table, it follows the setting in the Datum Preference tab. The default is to Convert tables to NAD83.

Your table is geocoded with a NAD83 datum if:

- it has a metadata datum key of NAD83 or map file in NAD83 and you do not specify to convert it to NAD27; or
- you choose to convert it to NAD83 by clicking Yes at the pre-scan dialog box or accepting that setting in the Datum Preferences dialog box; or
- your table is in a coordinate system that is neither long/lat NAD83 nor NAD27.

Note: Points are created in the original system. Long/lat values are returned in NAD83. Examples of these tables are long/lat datum less or State Plane system.

Your table is geocoded with a NAD27 datum if:

- it has a metadata key of NAD27 or map file in NAD27 and the Datum Preferences dialog box is set to convert to NAD27; or
- you choose to convert it to NAD27 by clicking Yes at the pre-scan dialog box or accept that setting in the Datum Preferences dialog box.

To view the metadata for your table, open the .tab file in a text editor and look under the section called begin_metadata. The datum key is listed as \MapMarker\Preferences\Datum with possible values of NAD83 and NAD27. Setting datum preferences is discussed in Appendix B: MapMarker Preferences.

A general rule for TIGER-based data products is TIGER 90, 92 and 94 use NAD27 and TIGER 98 and 2000 use NAD83.
Datums and ODBC Tables

MapMarker is unable to store metadata in ODBC tables so MapMarker determines the datum from the projection in the MapInfo Map Catalog.
This section includes instructions for creating a MapInfo Map Catalog by hand and making a remote table mappable, two procedures that are necessary for geocoding remote tables. This information is designed for users who do not have access to MapInfo Professional. Users of MapInfo Professional are directed to Chapter 23 in the MapInfo Professional User’s Guide for instructions to carry out these steps automatically.

For complete details on remote table geocoding see Geocoding Remote Tables on page 93.

In this appendix:

- Manually Creating a MapInfo Map Catalog .................................................. 380
- Making a Remote Table Mappable .............................................................. 381
Manually Creating a MapInfo Map Catalog

You or your database administrator must create one MapInfo Map Catalog for each database you wish to access in MapMarker.

1. If the RDBMS requires owners and users, create the user MAPINFO with the password MAPINFO in the specific database where the mappable tables are located.
2. Create the table MAPINFO_MAPCATALOG in the database. The Create Table statement needs to be equivalent to the following SQL Create Table statement:

   ```sql
   Create Table MAPINFO_MAPCATALOG ( 
   SPATIALTYPE Float, 
   TABLENAME Char(32), 
   OWNERNAME Char(32), 
   SPATIALCOLUMN Char(32), 
   DB_X_LL Float, 
   DB_Y_LL Float, 
   DB_X_UR Float, 
   DB_Y_UR Float, 
   COORDINATESYSTEM Char(254), 
   SYMBOL Char(254), 
   XCOLUMNNAME Char(32), 
   YCOLUMNNAME Char(32), 
   RENDITIONTYPE Integer, 
   RENDITIONCOLUMN VarChar(32), 
   RENDITIONTABLE VarChar(32) 
   NUMBER_ROWS Integer
   )
   ```

   It is important that the structure of the table is exactly like this statement. The only substitution that can be made is for databases that support varchar or text data types; these data types can be substituted for the Char data type.

3. Create a unique index on the TABLENAME and OWNERNAME. For RDBMS’s that do not support owners, a unique table name is required. For other database systems, the Map Catalog requires a unique owner.tablename.

4. Grant Select privileges to all users on the MAPINFO_MAPCATALOG. This allows users to make tables mappable. The Update, Insert, and Delete privileges should be granted at the discretion of the database administrator.
Making a Remote Table Mappable

For each spatial table in the remote database that you want to access in MapMarker, you must add a row to the MAPINFO_MAPCATALOG table. This is carried out in MapInfo Professional when you choose **TABLE>MAINTENANCE>MAKE ODBC TABLE MAPPABLE**.

If you do not use MapInfo Professional to manage the Map Catalog, you must manually add rows to the MAPINFO_MAPCATALOG table for each spatial table in the database that you want to geocode. Each entry must contain the following information about the table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Values to Assign</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPATIALTYPE</td>
<td>4.0 for X,Y spatial index tables (Support for additional spatial servers is under development)</td>
<td>4.0</td>
</tr>
<tr>
<td>TABLENAME</td>
<td>Name of the table.</td>
<td>Drainage</td>
</tr>
<tr>
<td>OWNERNAME</td>
<td>Owner name.</td>
<td>Georgetown</td>
</tr>
<tr>
<td>SPATIALCOLUMN</td>
<td>Name of the column, if any containing spatial features. The name is:</td>
<td>NO_COLUMN</td>
</tr>
<tr>
<td></td>
<td>• NO_COLUMN (for mappable tables using X,Y)</td>
<td></td>
</tr>
<tr>
<td>DB_X_LL</td>
<td>X coordinate of the lower left corner of the layer’s bounding rectangle, in units indicated by the COORDINATESYSTEM as defined by MapInfo Professional (see below).</td>
<td>-360</td>
</tr>
<tr>
<td>DB_Y_LL</td>
<td>Lower left bounding Y value.</td>
<td>-90</td>
</tr>
<tr>
<td>DB_X_UR</td>
<td>Upper right bounding X value.</td>
<td>360</td>
</tr>
<tr>
<td>DB_Y_UR</td>
<td>Upper right bounding Y value.</td>
<td>90</td>
</tr>
<tr>
<td>COORDINATESYSTEM</td>
<td>A string representing a MapInfo-supported coordinate system that specifies a map projection, coordinate units, etc. Values are one of:</td>
<td>Earth Projection 1,0</td>
</tr>
<tr>
<td></td>
<td>• Earth Projection 1,0 (for NAD27)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Earth Projection 1,62 (for NAD27)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Earth Projection 1,33 (for NAD 83)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Earth Projection 1,74 (for NAD 83)</td>
<td></td>
</tr>
<tr>
<td>SYMBOL</td>
<td>A MapInfo Symbol clause (for a layer containing points)</td>
<td>Symbol (35,0,12)</td>
</tr>
<tr>
<td>XCOLUMNNAME</td>
<td>Specify the name of the column containing X coordinates.</td>
<td>NO_COLUMN</td>
</tr>
<tr>
<td>YCOLUMNNAME</td>
<td>Specify the name of the column containing Y coordinates.</td>
<td>NO_COLUMN</td>
</tr>
<tr>
<td>Column Name</td>
<td>Values to Assign</td>
<td>Example</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>RENDITIONTYPE</td>
<td>Specify 1 if on, 0 if off.</td>
<td>1</td>
</tr>
<tr>
<td>RENDITIONCOLUMN</td>
<td>Specify the name of the rendition column.</td>
<td>MI_SYMBOLOGY</td>
</tr>
<tr>
<td>RENDITIONTABLE</td>
<td>Specify the name of the rendition table. This field is not currently used and is</td>
<td>left empty</td>
</tr>
<tr>
<td></td>
<td>in place for future enhancements.</td>
<td></td>
</tr>
<tr>
<td>NUMBER_ROWS</td>
<td>Specify the number of rows in the table.</td>
<td>11</td>
</tr>
</tbody>
</table>
In this FAQ:

- Q: I have a table that I cannot geocode because I get the error “Unable to update row.” What does this mean and how can I fix it?
- Q: I open a table in MapInfo Professional and I am able to open that same table in MapMarker. Is this safe?
- Q: How can I improve MapMarker's performance?
- Q: What determines if a row is unmatched?
- Q: After geocoding, some of my records have a result code of “ND.” What does this mean?
- Q: What is the breakdown of the Census Block code?
- Q: What is the MapMarker Geoengine API and how can I use it?
- Q: When geocoding, how can I prevent firm names from matching to street names?
- Q: Why am I generating different results from the GeoEngine APIs vs. the MapMarker Desktop (UI)?
Q: I have a table that I cannot geocode because I get the error “Unable to update row.” What does this mean and how can I fix it?

A: This error occurs when MapMarker attempts to create the point object for a given row and is unable to do so. To fix this problem, remove the map from the table, and use either MapMarker or MapInfo Professional to re-create the map file associated with the table.

To use MapInfo Professional to remove the map file association, select TABLE > MAINTENANCE > TABLE STRUCTURE from the menu, and clear the “Table is Mappable” check box. This causes the map to be dropped from the table and all the objects associated with the table to be removed. Alternatively, you can delete the .MAP file associated with the table manually.

After dropping the map, simply open the table with MapMarker and re-geocode the table. MapMarker automatically makes any table that you open mappable, and you are able to geocode the table without getting the error.

Note that this error may also result from running out of disk space while geocoding with MapMarker.

Q: I open a table in MapInfo Professional and I am able to open that same table in MapMarker. Is this safe?

A: No, you should take care to avoid opening tables with MapMarker that are already open in MapInfo Professional. Because of limitations on the number of files that a Windows application can open at one time, MapInfo Professional closes files that are not actively being used. Once MapInfo Professional has closed all the files associated with a table, MapMarker is able to open that same table, potentially causing a conflict between the two applications.

Q: How can I improve MapMarker’s performance?

A: Performance is improved by sorting the table to geocode by ZIP Codes. You can also improve performance by requiring an exact match on street names and/or ZIP Codes.

The biggest improvements, however, comes from optimizing the performance of the disk where you store the matching dictionary. For best performance, copy the entire address dictionary to a local or network hard disk and set up a high performance disk cache. If storing the entire address dictionary (approx. 1 gigabyte for nationwide data) on a hard disk is not possible, copy just the configuration files to your hard disk (see the Getting Started Guide for instructions), and make sure that you have a disk cache set up for your CD-ROM drive.

Q: What determines if a row is unmatched?

A: Unmatched rows are those rows in your table that have a result code that is either blank or contains an “N”. These are the only rows that MapMarker attempts to match when you choose to geocode unmatched rows. It is important to note that unlike the MapInfo geocoder, and the EasyMarker application, MapMarker does not use the existence of a point object to determine if a row has been previously matched.

You should also note that MapMarker does not remove points from table rows that it does not match. If a table contains pre-existing point objects, delete these objects before geocoding to ensure that your results accurately reflect your current geocoding session.
If you have records for which you want to preserve pre-existing points, mark them as non-geocodable by putting the result code "NG" into the result code column. MapMarker does not attempt to match records with a result code of “NG” even if you choose to match all rows in a table.

**Q: After geocoding, some of my records have a result code of “ND.” What does this mean?**

**A:** ND stands for “No Data.” This means that the file containing the address matching data needed for that record was not found in your data path or that the area is not licensed. You get this result if you copy a subset of the MapMarker address dictionary to your hard disk and attempt to geocode addresses that are outside of that area.

Remember, you can include more than one directory in your MapMarker data path, and the path is searched in the order in which it appears. This means that you can move the two-digit ZIP Code files for New England to your hard drive and leave the complete U.S. Address Dictionary on CD-ROM. Your data path would be:

```
c:\mapinfo\mapmarkr\nedata;d:\data
```

where `c:\mapinfo\mapmarkr\nedata` is the hard drive location of your New England data subset and `d:\data` is the Address Dictionary CD-ROM. The path components are separated by a semicolon.

See the Installation chapter in the Getting Started guide, for more information about moving data subsets.

**Q: What is the breakdown of the Census Block code?**

**A:** MapMarker can return the full 15-digit Census Block ID only for a successfully geocoded record with an S5 result code. An S4 does not return any census code because it needs a house number to match. An S3 result does not qualify for the full code since it represents an address at a ZIP+4 location, which may cover more than one Census Block. An S3 result can return no more than a 12-digit Census Block Group number.

The complete Census Block ID code is represented as follows:

```
SSCCCTTTTTTGBB(B)
```

where

- **S** = State FIPS (Federal Information Processing Standard) code (2 characters)
- **C** = County FIPS code (3 characters)
- **T** = Census Tract (6 characters)
- **G** = Census Block Group (1 character)
- **B** = Census Block (typically 2 characters, sometimes 3).

**Q: What is the MapMarker Geoengine API and how can I use it?**

**A:** The MapMarker Geoengine API is a programmatic interface to the 32 bit Windows DLLs that contains the code that MapMarker uses to match addresses and return the lat/long coordinates of those addresses (UNIX libraries are also available). We provide this API and the header files that allow you to use it so that you can add geocoding capabilities to your own programs.
Included are C header files, definition files for MapBasic and Visual Basic, and several sample programs that demonstrate how to use the geocoding libraries. Note that use of the MapMarker API is not limited to these development environments only. The API can be used with any environment that provides a way to call routines in an external DLL.

For more on the API, see Part II of the MapMarker Product Guide.

Q: When geocoding, how can I prevent firm names from matching to street names?
A: One way to keep firm names from matching to street names is to require exact match on the house number. MapMarker does not match an input without a house number to an address with a house number. A firm name never has a house number so you never get a firm name matching to a street range.

Data often has the firm names and street names in the same columns or swapped. Therefore, MapMarker looks at both the firm name and the street address fields to try to determine the address.

Firm names can be very close to a street names. For example, "Little Johns", "Albany NY 12203" could match to "Little Johns Pizza" or "1 Little John Rd" which are both in that ZIP Code. Neither one is an exact match to the input. In this case it would return a multiple match.

In general, 99% of the time you should require an exact match on the house number. Not requiring this does introduce a significant chance of positioning the point a distance from the real point. If MapMarker does not find the house number in a range for a street, it matches it to the closest one. It might be helpful, however, to change some portion of the street address. For example, if you gave an address of "6500 West Peachtree St" it should resolve to "6500 Peachtree Industrial" instead of resolving to "1877 West Peachtree."

Q: Why am I generating different results from the GeoEngine APIs vs. the MapMarker Desktop (UI)?
A: When geocoding using the GeoEngine APIs and the MapMarker Desktop (UI) there are differences between both result codes and candidates.

ResultCode/GeoResult – If the GeoEngBuildResultCode() is used from the API, then result codes should be the same except that there is no ND. These are just Ns.

Candidates – If comparing table geocodes from batch geocoding to geocodes running the API in an application against the same table, the results should be the same (assuming the same geocoding preferences were used). Differences may occur when handling addresses with PMB, since PMB handling exists in the GUI CASS mode. These results, therefore, should not be compared to non-CASS mode results.

QuickFind/Interactive candidate lists can be different than candidate lists generated from the API. Both methods relax streetnames, postcode, and house number. In addition, they both set the interactive flag (GeoEngSetInteractive()). This flag can dramatically affect the candidate list returned. Setting this flag allows the maximum number of candidates to be returned. Candidates that would ordinarily be discarded are returned in this mode.
MapMarker Utility Programs

This section describes two utility programs that are installed with MapMarker: Find Address and Append. Note that these are .mbx programs that must be run from MapInfo Professional.

In this appendix:

- FindAddress .............................................................. 388
- Append ................................................................. 389
FindAddress is a MapBasic application that allows you to find any address contained in the MapMarker Address Dictionary or from a customized user dictionary and display the location in MapInfo Professional. You can find street intersections as well. FindAddress is included on the MapMarker CD-ROM and installs in the c:\mapinfo\mapmarker directory. FindAddress uses the MapMarker geocoding engine and the Address Dictionary to match the U.S. address or street intersection you want to find and spot on the map.

FindAddress is run from within MapInfo Professional. It is installed on the Tools menu when you invoke the MapBasic application.

Running FindAddress

To run FindAddress:

1. From MapInfo Professional, choose Run MapBasic Program. The Run MapBasic Program dialog box is displayed.
2. Choose findad32.mbx from the MapMarker directory. Click OK. FindAddress is added to the Tools menu.
3. With an active Map window displaying the general area of the address you wish to find, choose Tools>FindAddress>Locate Address. The Find Address dialog box is displayed.
4. Type the address you want to find. You must enter the street name and either the city/state or the ZIP Code. Click on the FindAddress button. MapInfo Professional locates the address and marks the location with a symbol. To match to ZIP Code centroids, enter at least a 5 or 9 digit ZIP Code and click on the Map Centroid button.

   To find a street intersection, in the Street Address text box enter the first street name followed by the ampersand (&) or the word and followed by the second street name. For example: East 37th Street & Avenue P. Click on the FindAddress button. MapInfo Professional locates the intersection.

5. If multiple address or intersection candidates are found, FindAddress displays them in the Candidate Address dialog box. Highlight the appropriate address or intersection. The ranges of that address are displayed in the Ranges dialog box. To spot the location on the map, click the Map Candidate button.
6. To find the ZIP Code centroid for your address, type in the ZIP Code and click the Map Centroid button. MapInfo Professional spots a symbol at the ZIP Code centroid.
7. Close the FindAddress dialog box by clicking on the Close Window button. The Map window is now active and the symbol for the located address is visible.
8. To change the default symbol, the zoom level for the Map window, or to relax address matching conditions, choose Tools>FindAddress>Options. Note that changing the zoom level in the Options dialog box only takes effect after you have located another address in the FindAddress dialog box. For the zoom level units, FindAddress uses the settings you set in Map>Map Options.
Address Dictionary

FindAddress needs to know the location of the MapMarker Address Dictionary in order to find your addresses. This information is contained in two places in the Windows Registry:

- HKEY_LOCAL_MACHINE\SOFTWARE\MAPINFO\MAPMARKER\<version number>\SYSTEM\DatabasePath
- HKEY_CURRENT_USER\SOFTWARE\MAPINFO\MAPMARKER\<version number>\SYSTEM\DatabasePath

**Note:** In references to the MapMarker registry keys, `<version number>` indicates the major-release version of MapMarker, e.g., 10.0, even if you are running one of the point releases of that version, e.g., 10.1, 10.2.

If you move the Address Dictionary at any time, be sure to update the new path information in either of these places so that FindAddress works properly.

You can find addresses or street intersections from a customized user dictionary as well as from the MapMarker Address Dictionary. See Chapter 8: User Dictionaries, for more information.

Using FindAddress with StreetWorks

If you are locating addresses with FindAddress and are using StreetWorks display streets maps as a reference for the address, you may find rare instances where the address does not correspond to the underlying StreetWorks street. This is likely due to the elimination of some information about the street, such as a duplicate name. This behavior supports creating smaller StreetWorks files.

Additionally, the StreetWorks data was originally created from county-wide StreetInfo files which sometimes had data deleted (e.g., address ranges) in order to prevent address matching conflicts. FindAddress is based on a more advanced algorithm and does not delete data.

Append

Append is a utility program included in MapMarker that combines two or more MapInfo tables into a new table or adds them to an existing table. These tables must have the same number of columns and data types for each column. We recommend using Append for making one large MapInfo table when creating MapMarker User Dictionaries.

Append runs from within MapInfo Professional as an add-on application. A series of dialog boxes are displayed where you can specify the starting drive and directory of the search and select any available options. Append.mbx is located in the mbx directory after installing MapMarker.

If you are running the 16-bit version of MapInfo Professional (3.x or 4.x), Append uses two files, append.mbx and iputil16.dll. These two files must be in the same directory. For users of 32-bit MapInfo Professional (4.x, 5.x, 6.x) append.mbx and iputil32.dll must be in the same directory.
Running Append

To run the Append program from within MapInfo:

1. In MapInfo Professional, choose **FILE>RUN MAPBASIC PROGRAM**. The **RUN MAPBASIC PROGRAM** dialog box is displayed.
2. Choose **APPEND.MBX** from the directory where MapMarker is installed.
3. Click **OK**. The DataKit menu is added to the menu bar.
4. Choose **DATAKIT>APPEND>APPEND TABLES** to display the **APPEND TABLES** dialog box.
5. Choose a processing mode. Select **USE INTERACTIVE MODE** to choose individual groups of tables to append. The tables are placed in a queue for Append to concatenate. If you want Append to use a text file to determine what tables it should append and how it should append them, select the **USE INPUT FILE MODE** check box. For more instructions on Interactive Mode and Input File Mode see the following sections.
6. Click **OK** in the **APPEND TABLES** dialog box to start processing.

While Append runs, the program appends tables and writes out the new appended table with or without the table ID field, depending on the options you have chosen. When the program finishes, Append records a summary of the results in the message file and a dialog is displayed showing the elapsed processing time. You may open the message files in any text editor such as Notepad.

7. When you have finished, choose **DATAKIT>APPEND>EXIT APPEND** to close the program. If the Append command is the last item left in the DataKit menu, exiting the program also removes the DataKit menu.

Interactive Mode

Interactive Mode allows you to specify each process one at a time. This allows you to build an interactive list of tables that Append processes. Follow these guidelines to use Append in interactive mode. Follow these steps for each group of tables you wish to append.

1. Click on the tables you want to select in the **TABLE NAMES** list. You can change the directory and drive to display other tables. Keep the following in mind when choosing your tables:
   - Append does not let you concatenate tables if the numbers of columns in all of the tables are not the same. The program returns you to the **APPEND TABLES** dialog box if you attempt to do this.
   - If the number of columns in all of the tables is the same but they do not share the same data types between column numbers, Append tries to resolve the differences in the column types. When this happens, Append may change some of the data types of your columns. If there are changes, Append records these changes in the message file. If Append cannot resolve the data type differences, the program returns you to the **APPEND TABLES** dialog box.
   - If the number and types of columns in the appended tables are the same but the names are different, Append names the resulting columns according to the following rules. If you are appending tables to an existing table, the column names of the existing table does not change. If you are creating a new table with Append, the table has the same name as the first table that you selected in the **TABLE NAMES** list.
2. To insert a table identifier field into the output table, select the Insert Table ID field check box. The table identifier field contains the source table name for each row.
3. Click on **ADD TO LIST**. Append prompts you for the output name of the appended table. Enter the table name and click **OK**.

4. Click on **VIEW/SAVE LIST** to inspect or save the current interactive list. You may want to do this if you intend to process the same tables later in **Input File Mode**. To remove all the items in the current interactive list, click **CLEAR LIST**.

**Input File Mode**

Input File Mode allows you to specify an input text file that Append uses to process many tables without having to add them into a list one at a time. The input file is a formatted file that contains command lines for Append. The command lines list the files that are joined and indicates to Append what options to use when processing. This method is most convenient when you have a large number of tables that must be appended and you do not want to go through the tedious process of selecting the tables through the graphical interface.

Below is the format of the command line that you must use and some examples that use the format.

```plaintext
append <input_table_name_list> [/table-id] /output <output_table_name>
```

e.g.,

```plaintext
append c:\data\test1.tab test2.tab /table-id /output newtest.tab
append test1.tab test2.tab /output c:\data\newtest.tab
```

The first example creates a table called `newtest` that is a concatenation of two other tables: `test1` and `test2`. This example also creates a new column in the newtest table (Table_ID) that holds the source table for each record.

The second example is similar to the first except that Append does not create a Table_ID column; the paths for the source and output tables are also different.

To use Append in **Input File Mode**, do the following:

1. Select the **Use Input File Mode** check box.
2. Click the **SET INPUT FILE NAME** button to specify the name of the input file that Append should use.

**The APPEND.MSG File**

Append produces a message file for each processing session that contains information about the tables that were appended. Append names the message file `APPEND.MSG` and places the file in the same directory as the output tables. If the file `APPEND.MSG` already exists, Append adds the new information to the end of the existing file.

**Considerations When Running Append**

If you append a file that is currently open, Append closes the table before the program processes it. If the table has pending edits, Append displays a dialog box that asks you whether you want to save or discard those edits before appending.

If you are using input file mode, be sure that none of your command lines conflict or overwrite files produced by a previous line in your input file.
MapMarker installs the following files:

<table>
<thead>
<tr>
<th>Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapmarkr.exe</td>
<td>Program file</td>
</tr>
<tr>
<td>mmserve.exe</td>
<td>MapMarker Server</td>
</tr>
<tr>
<td>mapmarkr.ocx</td>
<td>MapMarker Geocoder Control</td>
</tr>
<tr>
<td>mm32v10.dll</td>
<td>Geocoder Engine</td>
</tr>
<tr>
<td>mm32v5.dll</td>
<td>Provided for backward compatibility for version 5.x, 6.x, 7.x, 8.x, and 9.x geocoding applications</td>
</tr>
<tr>
<td>mm32v6.dll</td>
<td></td>
</tr>
<tr>
<td>mm32v7.dll</td>
<td></td>
</tr>
<tr>
<td>mm32v8.dll</td>
<td></td>
</tr>
<tr>
<td>mm32v9.dll</td>
<td></td>
</tr>
<tr>
<td>mimfa750.dll</td>
<td>MapInfo File Access Library</td>
</tr>
<tr>
<td>MapMarker.chm</td>
<td>Online Help files</td>
</tr>
<tr>
<td>Mapmarker_10_Installer.chm</td>
<td>Online documentation files</td>
</tr>
<tr>
<td>MapmarkerPlus10.pdf</td>
<td></td>
</tr>
<tr>
<td>MapmarkerPlus10Installer.pdf</td>
<td></td>
</tr>
<tr>
<td>HKEY_LOCAL_MACHINE\SOFTWARE\MAPINFO\MAPMARKER\10.0&quot;</td>
<td>System setup defaults for 32-bit MapMarker</td>
</tr>
<tr>
<td>findad32.mbx</td>
<td>A MapBasic executable program for locating nationwide addresses</td>
</tr>
<tr>
<td>xxz9.cen</td>
<td>State centroid file for Address Dictionary:</td>
</tr>
<tr>
<td>xxzz.adr xxzz.adx</td>
<td>xx = state two-letter abbreviation</td>
</tr>
<tr>
<td></td>
<td>zz= two-digit ZIP Code</td>
</tr>
<tr>
<td></td>
<td>Also includes military ZIP Codes (e.g., mx0z.adr/adx)</td>
</tr>
<tr>
<td>Files</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>ecodes.*</td>
<td>Address Dictionary configuration files</td>
</tr>
<tr>
<td>usafin.cdb</td>
<td></td>
</tr>
<tr>
<td>airport.*</td>
<td></td>
</tr>
<tr>
<td>cityfin.*</td>
<td></td>
</tr>
<tr>
<td>cityline.*</td>
<td></td>
</tr>
<tr>
<td>cityurb.*</td>
<td></td>
</tr>
<tr>
<td>cityunz.*</td>
<td></td>
</tr>
<tr>
<td>cityunz4.*</td>
<td></td>
</tr>
<tr>
<td>cityzip.*</td>
<td></td>
</tr>
<tr>
<td>file.tbl</td>
<td></td>
</tr>
<tr>
<td>findctrd.*</td>
<td></td>
</tr>
<tr>
<td>geo_usa.*</td>
<td></td>
</tr>
<tr>
<td>hwyexits.*</td>
<td></td>
</tr>
<tr>
<td>pointzips.*</td>
<td></td>
</tr>
<tr>
<td>sinames.*</td>
<td></td>
</tr>
<tr>
<td>zipmastr.*</td>
<td></td>
</tr>
<tr>
<td>zipmove.*</td>
<td></td>
</tr>
<tr>
<td><em>.las/</em>.los</td>
<td></td>
</tr>
<tr>
<td>dph.hsa</td>
<td>DPV data files. These files will only be present if DPV has been purchased and installed.</td>
</tr>
<tr>
<td>dph.hsc</td>
<td></td>
</tr>
<tr>
<td>dph.hsf</td>
<td></td>
</tr>
<tr>
<td>Append.mbx</td>
<td>Utility and supporting library for appending tables together</td>
</tr>
<tr>
<td>iputil32.dll</td>
<td></td>
</tr>
<tr>
<td>datadict.dll</td>
<td>Component containing the User Dictionary creation APIs</td>
</tr>
<tr>
<td>mmodbc.dll</td>
<td>MapMarker library allowing ODBC interactivity</td>
</tr>
<tr>
<td>Us_addr.dbf/us_addr.tab</td>
<td>Sample U.S. addresses</td>
</tr>
<tr>
<td>Dc_addr.tab</td>
<td>Sample Washington, D.C. addresses</td>
</tr>
</tbody>
</table>

* To view the settings for the system, run regedit.exe, found in the Windows 98/2000/XP or Windows NT directory.
The following tables list messages generated by catastrophic internal errors, API call errors, errors in the input data (such as bad postal codes), and when the evaluation copy count is exceeded.

### Internal engine errors, input data errors, and API call errors

<table>
<thead>
<tr>
<th>Error #</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GEO_ENG_MALLOC_ERR</td>
<td>Memory allocation of internal structure failed</td>
</tr>
<tr>
<td>2</td>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>Memory overwrite detected</td>
</tr>
<tr>
<td>3</td>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>NULL input parameter or invalid parameter values</td>
</tr>
<tr>
<td>4</td>
<td>GEO_ENG_FILE_NOT_FOUND_ERR</td>
<td>Input file was not found</td>
</tr>
<tr>
<td>5</td>
<td>GEO_ENG_FILE_OPEN_ERR</td>
<td>Error opening initialization or license files</td>
</tr>
<tr>
<td>6</td>
<td>GEO_ENG_FILE_READ_ERR</td>
<td>Error reading from finance record from DB file</td>
</tr>
<tr>
<td>7</td>
<td>GEO_ENG_FILE_WRITE_ERR</td>
<td>Couldn’t close previous .ADX</td>
</tr>
<tr>
<td>8</td>
<td>GEO_ENG_INDEX_OPEN_ERR</td>
<td>Couldn’t open next .ADX</td>
</tr>
<tr>
<td>9</td>
<td>GEO_ENG_INDEX_ACCESS_ERR</td>
<td>Error while trying to read dictionary index</td>
</tr>
<tr>
<td>10</td>
<td>GEO_ENG_BAD_DATABASE_ERR</td>
<td>No longer used</td>
</tr>
<tr>
<td>11</td>
<td>GEO_ENG_END_OF_DATA</td>
<td>Premature end of data found</td>
</tr>
<tr>
<td>12</td>
<td>GEO_ENG_BAD_POSTAL_CODE</td>
<td>Postal code is not valid and city/state is not valid</td>
</tr>
<tr>
<td>13</td>
<td>GEO_ENG_UNINIT_DB_COMPONENT</td>
<td>Requested database component failed to initialize</td>
</tr>
<tr>
<td>14</td>
<td>GEO_ENG_BAD_INPUT_ADDRESS</td>
<td>Input address is invalid or missing fields</td>
</tr>
<tr>
<td>15</td>
<td>GEO_ENG_NO_DATAAVAILABLE</td>
<td>A dictionary for the input address was not available</td>
</tr>
<tr>
<td>16</td>
<td>GEO_ENG_INVALID_ACCESS_ERR</td>
<td>Attempting to access value that is not available</td>
</tr>
<tr>
<td>18</td>
<td>GEO_ENG_BAD_LICFILE_ERR</td>
<td>Invalid license file</td>
</tr>
<tr>
<td>19</td>
<td>GEO_ENG_EXCEEDED_LIMIT</td>
<td>Exceeded trial license limit</td>
</tr>
</tbody>
</table>
## MapMarker GeoEngine Error Codes

### Internal engine errors, input data errors, and API call errors

<table>
<thead>
<tr>
<th>Error #</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>GEO_ENG_SYSTEM_INVALID_HANDLE_REPORTED</td>
<td>Invalid RPC Server Handle</td>
</tr>
<tr>
<td>30</td>
<td>GEO_ENG_NO_APS</td>
<td>Invalid airport abbreviation was entered</td>
</tr>
<tr>
<td>31</td>
<td>GEO_ENG_LOW_LEVEL_AP_ERROR</td>
<td>Low level operation error</td>
</tr>
<tr>
<td>40</td>
<td>GEO_ENG_CANNOT_SET_DPV_WITHOUT_CASS</td>
<td>Attempted to set DPV mode without CASS mode being activated.</td>
</tr>
<tr>
<td>41</td>
<td>DPV_FILES_NOT_FOUND_OR_CORRUPTED</td>
<td>DPV files are not present, or they are corrupted.</td>
</tr>
<tr>
<td>42</td>
<td>DPV_IS_DISABLED</td>
<td>DPV feature has been disabled because of a usage violation. Please contact your vendor to re-enable this feature.</td>
</tr>
<tr>
<td>51</td>
<td>GEO_ENG_COORDS_NOT_AVAILABLE</td>
<td>Postal Centroid Coords not available</td>
</tr>
<tr>
<td>52</td>
<td>GEO_ENG_BAD_CAND_INDEX</td>
<td>Match candidate index too large</td>
</tr>
<tr>
<td>53</td>
<td>GEO_ENG_BAD_RANGE_INDEX</td>
<td>Street House Range Index too large</td>
</tr>
<tr>
<td>55</td>
<td>GEO_ENG_PO_BOX_ADDRESS</td>
<td>No longer used</td>
</tr>
<tr>
<td>57</td>
<td>GEO_ENG_RURAL_RTE_ADDRESS</td>
<td>No longer used</td>
</tr>
<tr>
<td>58</td>
<td>GEO_ENG_MULTIPLE_INSTANCE</td>
<td>Initialize has been called multiple times</td>
</tr>
<tr>
<td>59</td>
<td>GEO_ENG_BAD_INTERSECT_INDEX</td>
<td>Index to intersection index is out of range</td>
</tr>
<tr>
<td>60</td>
<td>GEO_ENG_BAD_INTERSECT_FORMAT</td>
<td>Couldn’t find street or intersection in input</td>
</tr>
<tr>
<td>61</td>
<td>GEO_ENG_INTERSECTION_ADDRESS</td>
<td>Specified address is an intersection</td>
</tr>
<tr>
<td>62</td>
<td>GEO_ENG_NO_LINE_POINTS</td>
<td>Segment polyline info contains no points</td>
</tr>
<tr>
<td>63</td>
<td>GEO_ENG_BAD_URBANIZER_FORMAT</td>
<td>No longer used</td>
</tr>
</tbody>
</table>

### Generic Error Codes

<table>
<thead>
<tr>
<th>Error #</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>307</td>
<td>GEO_ENG_PARSE_INIT_ERR</td>
<td>Parsing file (geo_usa.*) is corrupt or missing</td>
</tr>
<tr>
<td>308</td>
<td>GEO_ENG_PARSE_ENGINE_ERR</td>
<td>Parsing subsystem error</td>
</tr>
<tr>
<td>309</td>
<td>GEO_ENG_NO_STREET_ADDRESS</td>
<td>Street Address was missing</td>
</tr>
<tr>
<td>310</td>
<td>GEO_ENG_PARSE_STACK_OVERFLOW</td>
<td>Parser stack overflow</td>
</tr>
<tr>
<td>563</td>
<td>GEO_ENG_MATCH_INIT_ERR</td>
<td>No longer used</td>
</tr>
<tr>
<td>564</td>
<td>GEO_ENG_MATCH_ENGINE_ERR</td>
<td>No longer used</td>
</tr>
</tbody>
</table>
These errors are generated when problems occur when creating a custom user dictionary.

### User Dictionary Errors

<table>
<thead>
<tr>
<th>Error #</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>700</td>
<td>GEO_ENG_DD_ALLOCATE_ERR</td>
<td>Unable to allocate memory</td>
</tr>
<tr>
<td>701</td>
<td>GEO_ENG_DD_FIELD_DOES_NOT_EXIST_ERR</td>
<td>A specified field was not found in the attribution table</td>
</tr>
<tr>
<td>702</td>
<td>GEO_ENG_DD_END_ROW_TOO_LARGE_ERR</td>
<td>Final row number greater than total number of rows</td>
</tr>
<tr>
<td>703</td>
<td>GEO_ENG_DD_START_ROW_GT_END_ROW_ERR</td>
<td>Start row is greater than the end row</td>
</tr>
<tr>
<td>704</td>
<td>GEO_ENG_DD_START_ROW_NEGATIVE_ERR</td>
<td>Start row is less than 0</td>
</tr>
<tr>
<td>705</td>
<td>GEO_ENG_DD_INVALID_FIELD_NAME_ERR</td>
<td>Not Used any longer</td>
</tr>
<tr>
<td>706</td>
<td>GEO_ENG_DD_INVALID_UDINFO_ERR</td>
<td>Null pointer to User Dictionary info structure</td>
</tr>
<tr>
<td>707</td>
<td>GEO_ENG_DD_COPY_FIELD_ERR</td>
<td>No longer used</td>
</tr>
<tr>
<td>708</td>
<td>GEO_ENG_DD_TEMP_DIR_ERR</td>
<td>Unable to generate temp name.</td>
</tr>
<tr>
<td>709</td>
<td>GEO_ENG_DD_ADD_PATH_SLASH_ERR</td>
<td>No longer used</td>
</tr>
<tr>
<td>710</td>
<td>GEO_ENG_DD_GET_FIELD_ERR</td>
<td>Unable to find field</td>
</tr>
<tr>
<td>711</td>
<td>GEO_ENG_DD_REMOVE_FILE_ERR</td>
<td>Unable to remove UD temp file</td>
</tr>
<tr>
<td>712</td>
<td>GEO_ENG_DD_BAD_HANDLE_ERR</td>
<td>An required internal handle is NULL</td>
</tr>
<tr>
<td>713</td>
<td>GEO_ENG_DD_INVALID_OBJECT_ERR</td>
<td>Pointer to row object is NULL</td>
</tr>
<tr>
<td>714</td>
<td>GEO_ENG_DD_GET_CWD_ERR</td>
<td>Unable to get Current Working Directory</td>
</tr>
<tr>
<td>715</td>
<td>GEO_ENG_DD_SET_CWD_ERR</td>
<td>Unable to change directory</td>
</tr>
<tr>
<td>716</td>
<td>GEO_ENG_DD_BAD_PARAM_ERR</td>
<td>Empty UD name or path</td>
</tr>
<tr>
<td>717</td>
<td>GEO_ENG_DD_NAME_TOO_LONG_ERR</td>
<td>File Name length longer than eight characters</td>
</tr>
<tr>
<td>718</td>
<td>GEO_ENG_DD_INVALID_FILE_EXT_ERR</td>
<td>Invalid file extension found</td>
</tr>
<tr>
<td>719</td>
<td>GEO_ENG_DD_INVALID_PROCESS_CODE_ERR</td>
<td>Invalid Process flag</td>
</tr>
<tr>
<td>720</td>
<td>GEO_ENG_DD_TRANSLATE_TABLE_ERR</td>
<td>No longer used</td>
</tr>
<tr>
<td>721</td>
<td>GEO_ENG_DD_CLOSE_ALL_ERR</td>
<td>Error closing temp files</td>
</tr>
<tr>
<td>722</td>
<td>GEO_ENG_DD_RENAME_FILE_ERR</td>
<td>Unable to rename file</td>
</tr>
<tr>
<td>723</td>
<td>GEO_ENG_DD_DIR_DOES_NOT_EXIST_ERR</td>
<td>Not used any longer</td>
</tr>
</tbody>
</table>
## User Dictionary Errors

<table>
<thead>
<tr>
<th>Error #</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>724</td>
<td>GEO_ENG_DD_INVALID_PROJECTION_ERR</td>
<td>Table has unsupported coordinate system</td>
</tr>
<tr>
<td>725</td>
<td>GEO_ENG_DD_INVALID_DATUM_ERR</td>
<td>Unsupported datum for coordinate system</td>
</tr>
<tr>
<td>726</td>
<td>GEO_ENG_DD_BAD_PARAMETER_ERR</td>
<td>An input parameter is invalid</td>
</tr>
<tr>
<td>727</td>
<td>GEO_ENG_DD_INVALID_DICT_NAME_ERR</td>
<td>User Dictionary name already exists in directory</td>
</tr>
<tr>
<td>728</td>
<td>GEO_ENG_DD_EMPTY_FIELD_ERR</td>
<td>Empty field name specified</td>
</tr>
<tr>
<td>729</td>
<td>GEO_ENG_DD_DICT_NAME_TOO_LONG_ERR</td>
<td>User Dictionary name longer than eight characters</td>
</tr>
<tr>
<td>730</td>
<td>GEO_ENG_DD_DATUM_CONVERSION_ERR</td>
<td>Error converting between datums</td>
</tr>
<tr>
<td>731</td>
<td>GEO_ENG_DD_INPUT_OUTPUT_DIR_SAME_ERR</td>
<td>Input and Output directories are identical</td>
</tr>
<tr>
<td>732</td>
<td>GEO_ENG_DD_CREATE_LOG_FILE_ERR</td>
<td>Unable to open log file</td>
</tr>
<tr>
<td>733</td>
<td>GEO_ENG_DD_CLOSE_LOG_FILE_ERR</td>
<td>Unable to close log file</td>
</tr>
<tr>
<td>734</td>
<td>GEO_ENG_DD_WRITE_LOG_FILE_ERR</td>
<td>Unable to write to log file</td>
</tr>
<tr>
<td>800</td>
<td>GEO_ENG_DD_MI_OPEN_TABLE_ERR</td>
<td>Error opening table</td>
</tr>
<tr>
<td>801</td>
<td>GEO_ENG_DD_MI_GET_NUM_ROWS_ERR</td>
<td>Unable to get number of rows in table</td>
</tr>
<tr>
<td>802</td>
<td>GEO_ENG_DD_MI_GET_NUM_FIELDS_ERR</td>
<td>Unable to get number of fields in table</td>
</tr>
<tr>
<td>803</td>
<td>GEO_ENG_DD_MI_GET_ATTR_DEFS_ERR</td>
<td>Unable to get field attribute information</td>
</tr>
<tr>
<td>804</td>
<td>GEO_ENG_DD_MI_CLOSE_TABLE_ERR</td>
<td>Unable to close table</td>
</tr>
<tr>
<td>805</td>
<td>GEO_ENG_DD_MI_FETCH_ROW_ERR</td>
<td>Error fetching row from table</td>
</tr>
<tr>
<td>806</td>
<td>GEO_ENG_DD_MI_INIT_ERR</td>
<td>Unable to initialize MFAL</td>
</tr>
<tr>
<td>807</td>
<td>GEO_ENG_DD_MI_TERM_ERR</td>
<td>Unable to uninitialize MFAL</td>
</tr>
<tr>
<td>808</td>
<td>GEO_ENG_DD_MI_PREPARE_TABLE_ERR</td>
<td>Unable to prepare table for opening</td>
</tr>
<tr>
<td>809</td>
<td>GEO_ENG_DD_MI_GET_COORDSYS_ERR</td>
<td>Unable to get coordsys clause from input table</td>
</tr>
<tr>
<td>819</td>
<td>GEO_ENG_INTERP_HOUSE_ERR</td>
<td>Unable to interpret house number</td>
</tr>
<tr>
<td>820</td>
<td>GEO_ENG_INTERP_BAD_INSET</td>
<td>Invalid linear inset value</td>
</tr>
<tr>
<td>821</td>
<td>GEO_ENG_INTERP_BAD_SETBACK</td>
<td>Invalid perpendicular setback value</td>
</tr>
<tr>
<td>900</td>
<td>GEO_ENG_DD_CREATE_STREET_DBF_ERR</td>
<td>Unable to create street DBF file</td>
</tr>
<tr>
<td>901</td>
<td>GEO_ENG_DD_OPEN_STREET_DBF_ERR</td>
<td>Unable to open street DBF file</td>
</tr>
</tbody>
</table>
### User Dictionary Errors

<table>
<thead>
<tr>
<th>Error #</th>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>902</td>
<td>GEO_ENG_DD_CLOSE_STREET_DBF_ERR</td>
<td>Unable to close street DBF file</td>
</tr>
<tr>
<td>903</td>
<td>GEO_ENG_DD_INIT_STREET_DBF_ERR</td>
<td>Unable to initialize street DBF file</td>
</tr>
<tr>
<td>904</td>
<td>GEO_ENG_DD_CREATE_SEGMENT_DBF_ERR</td>
<td>Unable to create segment DBF file</td>
</tr>
<tr>
<td>905</td>
<td>GEO_ENG_DD_OPEN_SEGMENT_DBF_ERR</td>
<td>Unable to open segment DBF file</td>
</tr>
<tr>
<td>906</td>
<td>GEO_ENG_DD_CLOSE_SEGMENT_DBF_ERR</td>
<td>Unable to close segment DBF file</td>
</tr>
<tr>
<td>907</td>
<td>GEO_ENG_DD_INIT_SEGMENT_DBF_ERR</td>
<td>Unable to initialize segment DBF file</td>
</tr>
<tr>
<td>908</td>
<td>GEO_ENG_DD_CREATE_RANGE_DBF_ERR</td>
<td>Unable to create range DBF file</td>
</tr>
<tr>
<td>909</td>
<td>GEO_ENG_DD_OPEN_RANGE_DBF_ERR</td>
<td>Unable to open range DBF file</td>
</tr>
<tr>
<td>910</td>
<td>GEO_ENG_DD_CLOSE_RANGE_DBF_ERR</td>
<td>Unable to close range DBF file</td>
</tr>
<tr>
<td>911</td>
<td>GEO_ENG_DD_INIT_RANGE_DBF_ERR</td>
<td>Unable to initialize range DBF file</td>
</tr>
<tr>
<td>912</td>
<td>GEO_ENG_DD_CREATE_POINT_DBF_ERR</td>
<td>Unable to create point DBF file</td>
</tr>
<tr>
<td>913</td>
<td>GEO_ENG_DD_OPEN_POINT_DBF_ERR</td>
<td>Unable to open point DBF file</td>
</tr>
<tr>
<td>914</td>
<td>GEO_ENG_DD_CLOSE_POINT_DBF_ERR</td>
<td>Unable to close point DBF file</td>
</tr>
<tr>
<td>915</td>
<td>GEO_ENG_DD_INIT_POINT_DBF_ERR</td>
<td>Unable to initialize point DBF file</td>
</tr>
<tr>
<td>916</td>
<td>GEO_ENG_DD_CREATE_STREET2_DBF_ERR</td>
<td>Unable to create street2 DBF file</td>
</tr>
<tr>
<td>917</td>
<td>GEO_ENG_DD_OPEN_STREET2_DBF_ERR</td>
<td>Unable to open street2 DBF file</td>
</tr>
<tr>
<td>918</td>
<td>GEO_ENG_DD_CLOSE_STREET2_DBF_ERR</td>
<td>Unable to close street2 DBF file</td>
</tr>
<tr>
<td>919</td>
<td>GEO_ENG_DD_INIT_STREET2_DBF_ERR</td>
<td>Unable to initialize street2 DBF file</td>
</tr>
<tr>
<td>920</td>
<td>GEO_ENG_DD_CREATE_SEGMENT2_DBF_ERR</td>
<td>Unable to create segment2 DBF file</td>
</tr>
<tr>
<td>921</td>
<td>GEO_ENG_DD_OPEN_SEGMENT2_DBF_ERR</td>
<td>Unable to open segment2 DBF file</td>
</tr>
<tr>
<td>922</td>
<td>GEO_ENG_DD_CLOSE_SEGMENT2_DBF_ERR</td>
<td>Unable to close segment2 DBF file</td>
</tr>
<tr>
<td>923</td>
<td>GEO_ENG_DD_INIT_SEGMENT2_DBF_ERR</td>
<td>Unable to initialize segment2 DBF file</td>
</tr>
<tr>
<td>924</td>
<td>GEO_ENG_DD_CREATE_RANGE2_DBF_ERR</td>
<td>Unable to create range2 DBF file</td>
</tr>
<tr>
<td>925</td>
<td>GEO_ENG_DD_OPEN_RANGE2_DBF_ERR</td>
<td>Unable to open range2 DBF file</td>
</tr>
<tr>
<td>926</td>
<td>GEO_ENG_DD_CLOSE_RANGE2_DBF_ERR</td>
<td>Unable to close range2 DBF file</td>
</tr>
<tr>
<td>927</td>
<td>GEO_ENG_DD_INIT_RANGE2_DBF_ERR</td>
<td>Unable to initialize range2 DBF file</td>
</tr>
<tr>
<td>928</td>
<td>GEO_ENG_DD_CREATE_POINT2_DBF_ERR</td>
<td>Unable to create point2 DBF file</td>
</tr>
<tr>
<td>929</td>
<td>GEO_ENG_DD_OPEN_POINT2_DBF_ERR</td>
<td>Unable to open point2 DBF file</td>
</tr>
<tr>
<td>930</td>
<td>GEO_ENG_DD_CLOSE_POINT2_DBF_ERR</td>
<td>Unable to close point2 DBF file</td>
</tr>
<tr>
<td>Error #</td>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>931</td>
<td>GEO_ENG_DD_INIT_POINT2_DBF_ERR</td>
<td>Unable to initialize point2 DBF file</td>
</tr>
<tr>
<td>932</td>
<td>GEO_ENG_DD_WRITE_STREET2_DBF_ERR</td>
<td>Unable to write to street2 DBF</td>
</tr>
<tr>
<td>933</td>
<td>GEO_ENG_DD_WRITE_SEGMENT2_DBF_ERR</td>
<td>Unable to write to segment2 DBF file</td>
</tr>
<tr>
<td>934</td>
<td>GEO_ENG_DD_WRITE_RANGE2_DBF_ERR</td>
<td>Unable to write to range2 DBF file</td>
</tr>
<tr>
<td>935</td>
<td>GEO_ENG_DD_WRITE_POINT2_DBF_ERR</td>
<td>Unable to write to point2 DBF file</td>
</tr>
<tr>
<td>936</td>
<td>GEO_ENG_DD_COPY_STREET_REC_ERR</td>
<td>Unable to copy street record</td>
</tr>
<tr>
<td>937</td>
<td>GEO_ENG_DD_COPY_SEGMENT_REC_ERR</td>
<td>Unable to copy segment record</td>
</tr>
<tr>
<td>938</td>
<td>GEO_ENG_DD_COPY_RANGE_REC_ERR</td>
<td>Unable to copy range record</td>
</tr>
<tr>
<td>939</td>
<td>GEO_ENG_DD_COPY_POINT_REC_ERR</td>
<td>Unable to copy point record</td>
</tr>
<tr>
<td>940</td>
<td>GEO_ENG_DD_CREATE_ZIPCODE_DBF_ERR</td>
<td>Unable to create zipcode DBF</td>
</tr>
<tr>
<td>941</td>
<td>GEO_ENG_DD_CLOSE_ZIPCODE_DBF_ERR</td>
<td>Unable to open zipcode DBF</td>
</tr>
<tr>
<td>942</td>
<td>GEO_ENG_DD_CREATE_STREET_DIR_DBF_ERR</td>
<td>Unable to create Street direction DBF</td>
</tr>
<tr>
<td>943</td>
<td>GEO_ENG_DD_CLOSE_STREET_DIR_DBF_ERR</td>
<td>Unable to create Street direction DBF</td>
</tr>
<tr>
<td>944</td>
<td>GEO_ENG_DD_CREATE_STREET_NAME_DBF_ERR</td>
<td>Unable to close Street name DBF</td>
</tr>
<tr>
<td>945</td>
<td>GEO_ENG_DD_CLOSE_STREET_NAME_DBF_ERR</td>
<td>Unable to close Street name DBF</td>
</tr>
<tr>
<td>946</td>
<td>GEO_ENG_DD_CREATE_STREET_TYPE_DBF_ERR</td>
<td>Unable to create Street type DBF</td>
</tr>
<tr>
<td>947</td>
<td>GEO_ENG_DD_CLOSE_STREET_TYPE_DBF_ERR</td>
<td>Unable to close Street type DBF</td>
</tr>
<tr>
<td>948</td>
<td>GEO_ENG_DD_CREATE_CITY_DBF_ERR</td>
<td>Unable to create City DBF</td>
</tr>
<tr>
<td>949</td>
<td>GEO_ENG_DD_OPEN_CITY_DBF_ERR</td>
<td>Unable to open City DBF</td>
</tr>
<tr>
<td>950</td>
<td>GEO_ENG_DD_CLOSE_CITY_DBF_ERR</td>
<td>Unable to close City DBF</td>
</tr>
<tr>
<td>951</td>
<td>GEO_ENG_DD_INIT_CITY_DBF_ERR</td>
<td>Unable to initialize City DBF</td>
</tr>
<tr>
<td>952</td>
<td>GEO_ENG_DD_CREATE_ZIP5INFO_DBF_ERR</td>
<td>Unable to create Zip5info DBF</td>
</tr>
<tr>
<td>953</td>
<td>GEO_ENG_DD_OPEN_ZIP5INFO_DBF_ERR</td>
<td>Unable to open Zip5info DBF</td>
</tr>
<tr>
<td>954</td>
<td>GEO_ENG_DD_CLOSE_ZIP5INFO_DBF_ERR</td>
<td>Unable to close Zip5info DBF</td>
</tr>
<tr>
<td>955</td>
<td>GEO_ENG_DD_INIT_ZIP5INFO_DBF_ERR</td>
<td>Unable to initialize Zip5info DBF</td>
</tr>
<tr>
<td>956</td>
<td>GEO_ENG_DD_WRITE_STREET_DBF_ERR</td>
<td>Unable to write to Street DBF</td>
</tr>
<tr>
<td>957</td>
<td>GEO_ENG_DD_WRITE_SEGMENT_DBF_ERR</td>
<td>Unable to write to Segment DBF</td>
</tr>
<tr>
<td>958</td>
<td>GEO_ENG_DD_WRITE_RANGE_DBF_ERR</td>
<td>Unable to write to Range DBF</td>
</tr>
<tr>
<td>959</td>
<td>GEO_ENG_DD_WRITE_POINT_DBF_ERR</td>
<td>Unable to write to Point DBF</td>
</tr>
</tbody>
</table>
These errors are generated when problems occur accessing the Address Dictionary, such as when files are missing or when an invalid serial number is used.
<table>
<thead>
<tr>
<th>Error #</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1075</td>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>Error accessing the Address Dictionary</td>
</tr>
<tr>
<td>1076</td>
<td>GEO_ENG_BAD_DB_INIT_FLAGS</td>
<td>Invalid DBTypes specified</td>
</tr>
<tr>
<td>1077</td>
<td>GEO_ENG_UNDEFINED_SEARCH</td>
<td>Trying to get next street without setting up the first</td>
</tr>
<tr>
<td>1078</td>
<td>GEO_ENG_MISSING_DATABASE_ERR</td>
<td>Data for street not licensed or not found</td>
</tr>
<tr>
<td>1079</td>
<td>GEO_ENG_DBQ_MISSING_LICFILE</td>
<td>License file not found</td>
</tr>
<tr>
<td>1080</td>
<td>GEO_ENG_DBQ_MISSING_FILEID_TABLE</td>
<td>File.tbl missing</td>
</tr>
<tr>
<td>1081</td>
<td>GEO_ENG_DBQ_MISSING_ADR_ADX</td>
<td>.adr/.adx files not found</td>
</tr>
<tr>
<td>1082</td>
<td>GEO_ENG_DBQ_BAD_HEADER</td>
<td>Unable to read the header of file.tbl</td>
</tr>
<tr>
<td>1083</td>
<td>GEO_ENG_DBQ_BAD_INDEX</td>
<td>Bad index to Z9 file</td>
</tr>
<tr>
<td>1084</td>
<td>GEO_ENG_BETA_EXPIRATION_ERR</td>
<td>Beta has expired</td>
</tr>
<tr>
<td>1085</td>
<td>GEO_ENG_INVALID_SERIAL_NUMBER</td>
<td>Invalid serial number</td>
</tr>
<tr>
<td>1086</td>
<td>GEO_ENG_DBQ_UDR_ADR_CONFLICT_ERR</td>
<td>Internal problem with flags defining UD or AD searching</td>
</tr>
<tr>
<td>1087</td>
<td>GEO_ENG_DBQ_USER_DICT_INIT_ERR</td>
<td>Error finding User Dictionary files</td>
</tr>
<tr>
<td>1088</td>
<td>GEO_ENG_DBQ_BAD_URBAN_INDEX</td>
<td>No longer used</td>
</tr>
<tr>
<td>1089</td>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_FILE</td>
<td>cityline.idx not found</td>
</tr>
<tr>
<td>1090</td>
<td>GEO_ENG_DBQ_CITY_LINE_INDEX_OPEN</td>
<td>could not open cityline.idx</td>
</tr>
<tr>
<td>1091</td>
<td>GEO_ENG_DBQ_LICFILE_EXPIRED</td>
<td>Trial license has expired</td>
</tr>
<tr>
<td>1092</td>
<td>GEO_ENG_MAX_USER_DICT_EXCEEDED_ERR</td>
<td>User has exceeded the User Dictionary limit (5).</td>
</tr>
<tr>
<td>1093</td>
<td>GEO_ENG_SINAMES_INDEX_FILE</td>
<td>Could not find sinames.idx</td>
</tr>
<tr>
<td>1094</td>
<td>GEO_ENG_SINAMES_INDEX_OPEN</td>
<td>Could not open sinames.idx</td>
</tr>
<tr>
<td>1095</td>
<td>GEO_ENG_Z9_CEN_FILE_NOT_FOUND_ERR</td>
<td>ZIP centroid (*.cen) files not found</td>
</tr>
<tr>
<td>1096</td>
<td>GEO_ENG_UNIQUE_ZIP_FILE</td>
<td>Could not find cityunz.cdb</td>
</tr>
<tr>
<td>1097</td>
<td>GEO_ENG_UNIQUE_ZIP_INDEX_FILE</td>
<td>Could not find cityunz.idx file</td>
</tr>
<tr>
<td>1098</td>
<td>GEO_ENG_UNIQUE_ZIP_INDEX_OPEN</td>
<td>Could not open cityunz.idx</td>
</tr>
<tr>
<td>1099</td>
<td>GEO_ENG_VALID_ADD_ON_INDEX_FILE</td>
<td>Error finding the cityunz4.idx file</td>
</tr>
<tr>
<td>1100</td>
<td>GEO_ENG_VALID_ADD_ON_INDEX_OPEN</td>
<td>Error opening the cityunz4.idx file</td>
</tr>
<tr>
<td>1101</td>
<td>GEO_ENG_URBANIZ_ZIP_INDEX_FILE</td>
<td>Could not find cityurb.idx</td>
</tr>
</tbody>
</table>
## Errors Accessing the Address Dictionary and License Files

<table>
<thead>
<tr>
<th>Error #</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1102</td>
<td>Could not open cityurb.idx</td>
</tr>
<tr>
<td>1103</td>
<td>No longer used</td>
</tr>
<tr>
<td>1104</td>
<td>Error finding the zipmove.idx file</td>
</tr>
<tr>
<td>1105</td>
<td>Error opening the zipmove.idx file</td>
</tr>
<tr>
<td>1106</td>
<td>Error finding the cityzip.idx</td>
</tr>
<tr>
<td>1107</td>
<td>Error opening the cityzip.idx</td>
</tr>
<tr>
<td>1228</td>
<td>Zipmastr.cdb or Zipmastr.idx not found</td>
</tr>
<tr>
<td>1230</td>
<td>Error reading the Zipmastr file</td>
</tr>
<tr>
<td>1232</td>
<td>Error opening the Zipmastr.idx file</td>
</tr>
<tr>
<td>1233</td>
<td>Error accessing zipmaster index</td>
</tr>
<tr>
<td>1334</td>
<td><em>.las/</em>.los files not found</td>
</tr>
<tr>
<td>1536</td>
<td>Error using JNI method</td>
</tr>
<tr>
<td>1113</td>
<td>The specified error code is not used</td>
</tr>
</tbody>
</table>
This appendix defines the registry entries that are created when MapMarker is installed. This applies to both MapMarker Plus and the standard MapMarker product.

It also provides a listing of files to include in your installation program when incorporating MapMarker geocoding functionality into custom applications.

## MapMarker Registry Entries

The following registry entries are created when the MapMarker executable is installed and registered via regsvr32.exe. These entries need to have information present before the Address Dictionary initializes.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\HKEY_LOCAL_MACHINE\software\mapinfo\mapmarker\&lt;version number&gt;\System\DatabasePath</code></td>
<td>Path to the Address Dictionary</td>
</tr>
<tr>
<td><code>\HKEY_LOCAL_MACHINE\software\mapinfo\mapmarker\&lt;version number&gt;\User\SerialNumber</code></td>
<td>Serial Number that corresponds to the license file.</td>
</tr>
</tbody>
</table>

* If using the OEM model, then the SerialNumber can be anything because the Serial Number is passed by the application. It is not read from the registry. OEM licenses are created using a program that is purchased separately from MapMarker.

**Note:** In the registry keys listed in the table, `<version number>` indicates the major-release version of MapMarker, e.g., 10.0, even if you are running one of the point releases of that version, e.g., 10.1, 10.2.
Required Files for Custom Applications

This section identifies the files to include in your installation program for custom geocoding applications. Four component scenarios are defined.

<table>
<thead>
<tr>
<th>Component</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapMarker OCX only</td>
<td>For calling the MapMarker Server over the Internet, network, or intranet.</td>
</tr>
<tr>
<td>MapMarker Server/OCX with Address Dictionary</td>
<td>For calling the MapMarker dll or MapMarker Server. Address Dictionary is installed on a local machine.</td>
</tr>
<tr>
<td>MapMarker OCX with Address Dictionary</td>
<td>For calling the MapMarker dll directly. Address Dictionary is installed on a local machine. This is available beginning with MapMarker 4.3.0 OCX.</td>
</tr>
<tr>
<td>MapMarker API with Address Dictionary</td>
<td>For developing custom applications. Address Dictionary is installed on a local machine.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Name</th>
<th>File Type</th>
<th>OCX Only</th>
<th>Server &amp; OCX with AD</th>
<th>OCX with AD</th>
<th>API with AD</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapmarkr.ocx</td>
<td>OCX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>mm32v10.dll</td>
<td>MapMarker engine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>mm_serve.exe</td>
<td>MapMarker Server</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>regsvr32.exe</td>
<td>Support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Alaska.las</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Alaska.los</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Conus.las</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Conus.los</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Hawaii.las</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Hawaii.los</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Prvi.las</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Prvi.los</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Stgeorge.las</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Stgeorge.los</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>File Name</td>
<td>File Type</td>
<td>OCX Only</td>
<td>OCX with AD</td>
<td>API with AD</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>----------</td>
<td>-------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Stlrnc.las</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Stlrnc.los</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Stpaul.las</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Stpaul.los</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>File.tbl</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Geo_usa.cls</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Geo_usa.dct</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Geo_usa.pat</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Geo_usa.exp</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Geo_usa.dr</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Geo_usa.stn</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Sinames.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Airport.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Airport.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityline.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityfin.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityfin.fdx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityline.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityurb.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityurb.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityunz.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityunz.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityunz4.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cityunz4.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>File Name</td>
<td>File Type</td>
<td>OCX Only</td>
<td>OCX &amp; OCX with AD</td>
<td>OCX with AD</td>
<td>API with AD</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------</td>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Cityzip.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cityzip.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ecodes.cdx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ecodes.dbf</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>highrise.cdx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>highrise.dbf</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>hwyexports.cdx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>hwyexports.dbf</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>pointzips.cdx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>pointzips.dbf</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sinames.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Usaфин.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Zipmastr.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Zipmastr.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Zipmove.cdb</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Zipmove.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Finctrd.idx</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mx0x.adr</td>
<td>Data</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mx0x.adx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mx9x.adr</td>
<td>Data</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mx9x.adx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State adr/adx files</td>
<td>Address Dictionary</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>State centroid files</td>
<td>Address Dictionary</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>License files for installed states</td>
<td>Initialization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OEM license files**</td>
<td>Licensing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
The MM0x.adr, MM0x.adx, MM9x.adr, and MM9.x.adx files are data files that are stored with every data option. They store data for:

- "GU" Guam
- "AP" Armed Forces Pacific
- "VI" Virgin Islands
- "FM" Federated States of Micronesia
- "AS" American Samoa
- "PW" Palau
- "MP" Mariana Islands
- "MH" Marshall Islands
- "AE" Armed Forces Europe
- "AA" Armed Forces Americas

* Example: ny*.adr, ny*.adx
† Example: nyz9.cen
‡ Example: natnplus.l30, nyplus.l30 for MapMarker Plus; nation.l30, ny.l30 for standard MapMarker
** Use one or the other licensing scheme. OEM licenses are created with a separately purchased program
This table identifies the range of ZIP Codes assigned to states and territories by the U.S. Postal Service.

<table>
<thead>
<tr>
<th>State/Territory</th>
<th>ZIP Code Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puerto Rico/Virgin Islands</td>
<td>00600-01000</td>
</tr>
<tr>
<td>Massachusetts</td>
<td>01000-02799 05500-05599</td>
</tr>
<tr>
<td>Rhode Island</td>
<td>02800-02999</td>
</tr>
<tr>
<td>New Hampshire</td>
<td>03000-03899 00200-00299</td>
</tr>
<tr>
<td>Maine</td>
<td>03900-04999</td>
</tr>
<tr>
<td>Vermont</td>
<td>05000-05999</td>
</tr>
<tr>
<td>Connecticut</td>
<td>06000-06999</td>
</tr>
<tr>
<td>New Jersey</td>
<td>07000-08999</td>
</tr>
<tr>
<td>New York</td>
<td>09000-14999 00400-06399</td>
</tr>
<tr>
<td>Pennsylvania</td>
<td>15000-19699</td>
</tr>
<tr>
<td>Delaware</td>
<td>19700-19999</td>
</tr>
<tr>
<td>District of Columbia</td>
<td>20000-20599</td>
</tr>
<tr>
<td>Maryland</td>
<td>20600-21999</td>
</tr>
<tr>
<td>Virginia</td>
<td>22000-24699 20100-20199</td>
</tr>
<tr>
<td>West Virginia</td>
<td>24700-26899</td>
</tr>
<tr>
<td>North Carolina</td>
<td>27000-28999</td>
</tr>
<tr>
<td>South Carolina</td>
<td>29000-29999</td>
</tr>
<tr>
<td>Georgia</td>
<td>30000-31999 39900-39999</td>
</tr>
<tr>
<td>Florida</td>
<td>32000-34999</td>
</tr>
<tr>
<td>State/Territory</td>
<td>ZIP Code Range</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Alabama</td>
<td>35000-36999</td>
</tr>
<tr>
<td>Tennessee</td>
<td>37000-38599</td>
</tr>
<tr>
<td>Mississippi</td>
<td>38600-39799</td>
</tr>
<tr>
<td>Kentucky</td>
<td>40000-42799</td>
</tr>
<tr>
<td></td>
<td>45200-45299</td>
</tr>
<tr>
<td>Ohio</td>
<td>43000-45999</td>
</tr>
<tr>
<td>Indiana</td>
<td>46000-47999</td>
</tr>
<tr>
<td>Michigan</td>
<td>48000-49999</td>
</tr>
<tr>
<td>Iowa</td>
<td>50000-52899</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>53000-54999</td>
</tr>
<tr>
<td>Minnesota</td>
<td>55000-56799</td>
</tr>
<tr>
<td>South Dakota</td>
<td>57000-57799</td>
</tr>
<tr>
<td>North Dakota</td>
<td>58000-58899</td>
</tr>
<tr>
<td>Montana</td>
<td>59000-59999</td>
</tr>
<tr>
<td>Illinois</td>
<td>60000-62999</td>
</tr>
<tr>
<td></td>
<td>75500-75599</td>
</tr>
<tr>
<td>Missouri</td>
<td>63000-65899</td>
</tr>
<tr>
<td>Kansas</td>
<td>66000-67999</td>
</tr>
<tr>
<td>Nebraska</td>
<td>68000-69399</td>
</tr>
<tr>
<td>Louisiana</td>
<td>70000-71499</td>
</tr>
<tr>
<td>Arkansas</td>
<td>71600-72999</td>
</tr>
<tr>
<td>Oklahoma</td>
<td>73000-74999</td>
</tr>
<tr>
<td>Texas</td>
<td>75000-79999</td>
</tr>
<tr>
<td></td>
<td>73300-73399</td>
</tr>
<tr>
<td></td>
<td>88500-88599</td>
</tr>
<tr>
<td>Colorado</td>
<td>80000-81699</td>
</tr>
<tr>
<td>Wyoming</td>
<td>82000-83199</td>
</tr>
<tr>
<td>Idaho</td>
<td>83200-83899</td>
</tr>
<tr>
<td>Utah</td>
<td>84000-84799</td>
</tr>
<tr>
<td>Arizona</td>
<td>85000-86599</td>
</tr>
<tr>
<td>New Mexico</td>
<td>87000-88499</td>
</tr>
<tr>
<td>State/Territory</td>
<td>ZIP Code Range</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Nevada</td>
<td>88900-89899</td>
</tr>
<tr>
<td>California</td>
<td>90000-96699</td>
</tr>
<tr>
<td>Hawaii</td>
<td>96700-96899</td>
</tr>
<tr>
<td>Oregon</td>
<td>97000-97999</td>
</tr>
<tr>
<td>Washington</td>
<td>98000-99499</td>
</tr>
<tr>
<td>Alaska</td>
<td>99500-99999</td>
</tr>
<tr>
<td>Guam</td>
<td>96900-96999</td>
</tr>
</tbody>
</table>
Street Suffix Abbreviations

The following codes are standard USPS abbreviations for words that are frequently used in street addresses.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Address Name Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aly</td>
<td>Alley</td>
</tr>
<tr>
<td>Anx</td>
<td>Annex</td>
</tr>
<tr>
<td>Arc</td>
<td>Arcade</td>
</tr>
<tr>
<td>Ave</td>
<td>Avenue</td>
</tr>
<tr>
<td>Bch</td>
<td>Beach</td>
</tr>
<tr>
<td>Bg</td>
<td>Burg</td>
</tr>
<tr>
<td>Blf</td>
<td>Bluff</td>
</tr>
<tr>
<td>Blvd</td>
<td>Boulevard</td>
</tr>
<tr>
<td>Bnd</td>
<td>Bend</td>
</tr>
<tr>
<td>Br</td>
<td>Branch</td>
</tr>
<tr>
<td>Brg</td>
<td>Bridge</td>
</tr>
<tr>
<td>Brk</td>
<td>Brook</td>
</tr>
<tr>
<td>Btm</td>
<td>Bottom</td>
</tr>
<tr>
<td>Byp</td>
<td>Bypass</td>
</tr>
<tr>
<td>Byu</td>
<td>Bayou</td>
</tr>
<tr>
<td>Cir</td>
<td>Circle</td>
</tr>
<tr>
<td>Clb</td>
<td>Club</td>
</tr>
<tr>
<td>Clfs</td>
<td>Cliffs</td>
</tr>
<tr>
<td>Cor</td>
<td>Corner</td>
</tr>
<tr>
<td>Cors</td>
<td>Corners</td>
</tr>
<tr>
<td>Cp</td>
<td>Camp</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Address Name Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Cpe</td>
<td>Cape</td>
</tr>
<tr>
<td>Cres</td>
<td>Crescent</td>
</tr>
<tr>
<td>Crk</td>
<td>Creek</td>
</tr>
<tr>
<td>Crse</td>
<td>Course</td>
</tr>
<tr>
<td>Cswy</td>
<td>Causeway</td>
</tr>
<tr>
<td>Ct</td>
<td>Court</td>
</tr>
<tr>
<td>Ctr</td>
<td>Center</td>
</tr>
<tr>
<td>Cts</td>
<td>Courts</td>
</tr>
<tr>
<td>Cv</td>
<td>Cove</td>
</tr>
<tr>
<td>Cyn</td>
<td>Canyon</td>
</tr>
<tr>
<td>DI</td>
<td>Dale</td>
</tr>
<tr>
<td>Dm</td>
<td>Dam</td>
</tr>
<tr>
<td>Dr</td>
<td>Drive</td>
</tr>
<tr>
<td>Dv</td>
<td>Divide</td>
</tr>
<tr>
<td>Est</td>
<td>Estates</td>
</tr>
<tr>
<td>Expy</td>
<td>Expressway</td>
</tr>
<tr>
<td>Ext</td>
<td>Extension</td>
</tr>
<tr>
<td>Fall</td>
<td>Fall</td>
</tr>
<tr>
<td>Fld</td>
<td>Field</td>
</tr>
<tr>
<td>Flds</td>
<td>Fields</td>
</tr>
<tr>
<td>Fls</td>
<td>Falls</td>
</tr>
<tr>
<td>Fit</td>
<td>Flats</td>
</tr>
<tr>
<td>Frd</td>
<td>Ford</td>
</tr>
<tr>
<td>Frg</td>
<td>Forge</td>
</tr>
<tr>
<td>Frk</td>
<td>Fork</td>
</tr>
<tr>
<td>Frks</td>
<td>Forks</td>
</tr>
<tr>
<td>Frst</td>
<td>Forest</td>
</tr>
<tr>
<td>Fry</td>
<td>Ferry</td>
</tr>
<tr>
<td>Ft</td>
<td>Fort</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Address Name Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Fwy</td>
<td>Freeway</td>
</tr>
<tr>
<td>Gdns</td>
<td>Gardens</td>
</tr>
<tr>
<td>Gln</td>
<td>Glen</td>
</tr>
<tr>
<td>Grn</td>
<td>Green</td>
</tr>
<tr>
<td>Grv</td>
<td>Grove</td>
</tr>
<tr>
<td>Gtwy</td>
<td>Gateway</td>
</tr>
<tr>
<td>Hbr</td>
<td>Harbor</td>
</tr>
<tr>
<td>Hl</td>
<td>Hill</td>
</tr>
<tr>
<td>Hls</td>
<td>Hills</td>
</tr>
<tr>
<td>Holw</td>
<td>Hollow</td>
</tr>
<tr>
<td>Hts</td>
<td>Heights</td>
</tr>
<tr>
<td>Hvn</td>
<td>Haven</td>
</tr>
<tr>
<td>Hwy</td>
<td>Highway</td>
</tr>
<tr>
<td>Inlt</td>
<td>Inlet</td>
</tr>
<tr>
<td>Is</td>
<td>Island</td>
</tr>
<tr>
<td>Isle</td>
<td>Isle</td>
</tr>
<tr>
<td>Iss</td>
<td>Islands</td>
</tr>
<tr>
<td>Jct</td>
<td>Junction</td>
</tr>
<tr>
<td>Knls</td>
<td>Knolls</td>
</tr>
<tr>
<td>Ky</td>
<td>Key</td>
</tr>
<tr>
<td>Lcks</td>
<td>Locks</td>
</tr>
<tr>
<td>Ldg</td>
<td>Lodge</td>
</tr>
<tr>
<td>Lf</td>
<td>Loaf</td>
</tr>
<tr>
<td>Lgt</td>
<td>Light</td>
</tr>
<tr>
<td>Lk</td>
<td>Lake</td>
</tr>
<tr>
<td>Lks</td>
<td>Lakes</td>
</tr>
<tr>
<td>Ln</td>
<td>Lane</td>
</tr>
<tr>
<td>Lndg</td>
<td>Landing</td>
</tr>
<tr>
<td>Loop</td>
<td>Loop</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Address Name Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Mall</td>
<td>Mall</td>
</tr>
<tr>
<td>Mdws</td>
<td>Meadows</td>
</tr>
<tr>
<td>Ml</td>
<td>Mill</td>
</tr>
<tr>
<td>Mls</td>
<td>Mills</td>
</tr>
<tr>
<td>Mnr</td>
<td>Manor</td>
</tr>
<tr>
<td>Msn</td>
<td>Mission</td>
</tr>
<tr>
<td>Mt</td>
<td>Mount</td>
</tr>
<tr>
<td>Mtn</td>
<td>Mountain</td>
</tr>
<tr>
<td>Nck</td>
<td>Neck</td>
</tr>
<tr>
<td>Orch</td>
<td>Orchard</td>
</tr>
<tr>
<td>Oval</td>
<td>Oval</td>
</tr>
<tr>
<td>Park</td>
<td>Park</td>
</tr>
<tr>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>Path</td>
<td>Path</td>
</tr>
<tr>
<td>Pike</td>
<td>Pike</td>
</tr>
<tr>
<td>Pky</td>
<td>Parkway</td>
</tr>
<tr>
<td>Pl</td>
<td>Place</td>
</tr>
<tr>
<td>Pln</td>
<td>Plain</td>
</tr>
<tr>
<td>Pins</td>
<td>Plains</td>
</tr>
<tr>
<td>Plz</td>
<td>Plaza</td>
</tr>
<tr>
<td>Pnes</td>
<td>Pines</td>
</tr>
<tr>
<td>Pr</td>
<td>Prairie</td>
</tr>
<tr>
<td>Prt</td>
<td>Port</td>
</tr>
<tr>
<td>Pt</td>
<td>Point</td>
</tr>
<tr>
<td>Radl</td>
<td>Radial</td>
</tr>
<tr>
<td>Rd</td>
<td>Road</td>
</tr>
<tr>
<td>Rdg</td>
<td>Ridge</td>
</tr>
<tr>
<td>Riv</td>
<td>River</td>
</tr>
<tr>
<td>Rnch</td>
<td>Ranch</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Address Name Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Row</td>
<td>Row</td>
</tr>
<tr>
<td>Rpds</td>
<td>Rapids</td>
</tr>
<tr>
<td>Rst</td>
<td>Rest</td>
</tr>
<tr>
<td>Run</td>
<td>Run</td>
</tr>
<tr>
<td>Shl</td>
<td>Shoal</td>
</tr>
<tr>
<td>Shls</td>
<td>Shoals</td>
</tr>
<tr>
<td>Shr</td>
<td>Shore</td>
</tr>
<tr>
<td>Shrs</td>
<td>Shores</td>
</tr>
<tr>
<td>Smt</td>
<td>Summit</td>
</tr>
<tr>
<td>Spg</td>
<td>Spring</td>
</tr>
<tr>
<td>Spgs</td>
<td>Springs</td>
</tr>
<tr>
<td>Spur</td>
<td>Spur</td>
</tr>
<tr>
<td>Sq</td>
<td>Square</td>
</tr>
<tr>
<td>St</td>
<td>Street</td>
</tr>
<tr>
<td>Sta</td>
<td>Station</td>
</tr>
<tr>
<td>Stra</td>
<td>Stravenue</td>
</tr>
<tr>
<td>Strm</td>
<td>Stream</td>
</tr>
<tr>
<td>Ter</td>
<td>Terrace</td>
</tr>
<tr>
<td>Tpke</td>
<td>Turnpike</td>
</tr>
<tr>
<td>Trak</td>
<td>Track</td>
</tr>
<tr>
<td>Trce</td>
<td>Trace</td>
</tr>
<tr>
<td>Tri</td>
<td>Trail</td>
</tr>
<tr>
<td>Trlr</td>
<td>Trailer</td>
</tr>
<tr>
<td>Tunl</td>
<td>Tunnel</td>
</tr>
<tr>
<td>Un</td>
<td>Union</td>
</tr>
<tr>
<td>Via</td>
<td>Viaduct</td>
</tr>
<tr>
<td>Vis</td>
<td>Vista</td>
</tr>
<tr>
<td>VI</td>
<td>Ville</td>
</tr>
<tr>
<td>Vlg</td>
<td>Village</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Address Name Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Vly</td>
<td>Valley</td>
</tr>
<tr>
<td>Vw</td>
<td>View</td>
</tr>
<tr>
<td>Walk</td>
<td>Walk</td>
</tr>
<tr>
<td>Way</td>
<td>Way</td>
</tr>
<tr>
<td>Wls</td>
<td>Wells</td>
</tr>
<tr>
<td>Xing</td>
<td>Crossing</td>
</tr>
</tbody>
</table>
Index

**A**

address columns
  select input columns 37, 39
  select input columns dialog 32
address dictionaries
  creating 109
  address dictionary 109
    creating 106
    creating subsets 28
    FindAddress utility 388–389
    installing 17
    NAD83 and NAD27 datums 376–377
    storing data 16
    storing on hard drive 16
    system preferences 371
address matching
  expanding search area 43
  geocoding 10
  input columns 32, 37, 39
  output columns 32, 35
  trade-offs 52
append.mbx 389, 391
applications
  creating 385
  files to include 406
applications, creating 266, 277
attribute 85–92
  adding attributes 89
    batch 90
    geocode 89
  attaching data to geocoded records 85, 91
  clear file selections 90
  column value 85, 89
  considerations 92
  output point 85, 88
  sample exercise 91
  stopping 90
automatic geocoding 40, 65

**B**

batch add attributes 90
batch geocoding 83–85
best match from multiple candidates, result codes 64
bind attribute table columns to output columns 88

**C**

CAddressList 171
candidate visualization
  displaying your match on a map 43
  interactive geocoding 57
  quick find 58
  street data path 57, 373
CASS certification
  geocoding for 70–71
  setting preferences 50
CD-ROM unlocking 113
census block ID 34
Census Block return values 385
CGeoOut object 200
clear file selections 90
ClearDialogText() 137
client applications 120
client/server geocoding 116
client/server toolkit
  client/server geocoding 116
  internet applications 127
  MapMarker OCX 120
  MapMarker server 116
column value attribution 85, 89
columns
  input 32, 37, 39
  output 32–33, 35
compatibility with earlier versions 28
Connect() 137–138
creating
  address dictionaries 106, 109
  map catalogs 95
  standalone geocoding applications 266, 277, 385
custom applications
  required files 406

**D**

data
  creating subsets 28
  moving 30
  storing 16
  using from CD-ROM 16
data unlocking 17
datums
  address dictionary 376–377
  choosing 377
  ODBC tables 378
  setting for remote tables 97–98
  system preferences 372
dBASE files, geocoding 78
Delivery Point Validation (DPV) 71–77
Disconnect() 138
displaying geocoded records in MapInfo 44
distance units 372
DoGeocode() 139
DoSetProperties() 139
DpvGeocodeAddress() 139–140, 232–233
DpvGeocodeAddressLastLine() 141–142, 234–235
DpvGeocodeAddressLastLineV() 205–206
DpvGeocodeAddressLastLineWithSerial() 142–143, 235, 237
DpvGeocodeAddressV() 204
DpvGeocodeAddressWithSerial() 143–144, 237–238

E
editing records
interactive geocoding 44
ingoing records, interactive geocoding 44
error codes 395–403
F
fallback settings 48
false positives
reducing 53–54
files
inclusion in custom applications 406
FindAddress 388–389
function calls
data dictionary API 340
MapMarker API 278–279
MapMarker server RPC API 231
G
GeocodeCandidateWithDpv() 255–256, 258
GeocodeErrorText() 258
GeocodeAddress() 145, 147, 239–242
GeocodeAddress_A() 187
GeocodeAddressEx() 147–148
GeocodeAddressEx_A() 188
GeocodeAddressExV() 208
GeocodeAddressLastLine() 148–149, 235, 242–243
GeocodeAddressLastLine_A() 189
GeocodeAddressLastLineEx() 150, 243–244
GeocodeAddressLastLineEx_A() 190
GeocodeAddressLastLineExV() 211
GeocodeAddressLastLineV() 210
GeocodeAddressLastLineWithSerial() 151, 245–246
GeocodeAddressLastLineWithSerialEx() 246–247
GeocodeAddressV() 207
GeocodeAddressWithSerial() 152–153, 248–249
GeocodeAddressWithSerial_A() 191, 193
GeocodeAddressWithSerialEx() 153, 249–250
GeocodeAddressWithSerialEx_A() 192
GeocodeCheckDbAvailability() 155, 251
GeocodeFreeSet() 155, 251
GeocodeFreeSet_A() 193
GeocodeFreeSetV() 212
GeocodeGetCandidate() 252–253
GeocodeGetCandidateCoords() 255
GeocodeGetCandidateEx() 253–254
GeocodeGetCandidates() 156–157
GeocodeGetCandidatesV() 212
GeocodeGetErrorText() 157, 258
GeocodeGetErrorText_A() 194
GeocodeGetErrorTextV() 213
GeocodeGetHwyExitCandidate() 158, 259
GeocodeGetHwyExitCandidate_A() 194
GeocodeGetHwyExitCandidateV() 213
GeocodeGetServerVersion() 159, 260
GeocodeGetServerVersionV() 215
GeocodeGetStatesFound() 159, 260
GeocodeGetStatesFound_A() 195
GeocodeGetStatesFoundV() 215
GeocodeGetStatesLicensed() 160, 260
GeocodeGetStatesLicensed_A() 196
GeocodeGetStatesLicensedV() 215
GeocodeHwyExit() 160, 261–262
GeocodeHwyExit_A() 196
GeocodeHwyExitV() 216
GeocodeIsCandidateMultiUnit() 161, 262
GeocodeIsCandidateMultiUnit_A() 198
GeocodeIsCandidateMultiUnitV() 217
GeocodePostalCentroid() 162–163, 263
GeocodePostalCentroid_A() 198
GeocodePostalCentroidV() 217–218
GeocodePostalCentroidWithSerial() 163–164, 264
GeocodePostalCentroidWithSerial_A() 199
GeocodeShutdown() 264
geocoding
client/server 116
place names 111
re-geocoding 45
geocoding applications
creating with MapMarker API 266, 277, 385
MapMarker geocoder control 120
geocoding from remote workstations 116, 124–125
geocoding matches
displaying your match on a map 43, 57–58
positional accuracy 54
strategies 53–54
to ZIP Code centroid 77
to zip code centroid 77
trade-offs 52
geocoding preferences 45, 47
geocoding procedures
  address columns 37, 39
  automatic 40, 65
  CASS certification 70–71
dBASE files 78
displaying geocoded records in MapInfo 44
  interactive 41, 43–44, 65
  one pass 55
  sorting tables 32
to zip code centroids 77
  using quick geocode 78
geocoding remote databases
  geocoding procedures 96, 98
  preparation procedures 95
  related table geocoding 98, 101
  remote mappable tables 94, 96
  rollback segment limit 103
  supported data types 103
  transaction control 102
geocoding results 65
  output columns 32, 35
  result codes 62, 66
  unmatched rows 384
GeoEngBrowseAirportByState() 280–281
GeoEngBrowsePlace() 282–283
GeoEngBrowseSetFilter() 283–284
GeoEngBrowseStreet() 284–285, 346
GeoEngBuildResultCode() 286
GeoEngCheckDbAvailability() 287, 289
GeoEngCheckDbAvailabilityWithLicense() 289, 291
GeoEngFindPostalCentroid() 291, 294
GeoEngGetAirport() 294, 296
GeoEngGetCandidateCoords() 297–298
GeoEngGetCandidateCount() 299
GeoEngGetCfgParms() 300, 302, 355
GeoEngGetErrorText() 302
GeoEngGetIndexedCandidate() 303, 305
GeoEngGetIndexedCoords() 306, 308
GeoEngGetIndexedHwyExit() 308
GeoEngGetNextHouseRange() 310–311, 356
GeoEngGetStatesFound() 311
GeoEngGetStatesLicensed() 312
GeoEngGetVersion() 313
GeoEngGetZIPCountyFIPSList() 313
GeoEngInit() 314, 316
GeoEngInitWithLicense() 317, 319
GeoEngIsCandidateMultiUnit() 319–320
GeoEngIsEval() 320
GeoEngKill() 321
GeoEngMatchAddress() 322, 325
GeoEngMatchFormattedAddress() 325, 328
GeoEngMatchHwyExit() 328
GeoEngNAD27ToNAD83() 330
GeoEngNAD83ToNAD27() 331
GeoEngParseLastLine() 332
GeoEngSetCfgParms() 333–334
GeoEngSetInteractive() 335
GeoEngInit() 314, 316
GeoEngInitWithLicense() 317, 319
GeoEngIsCandidateMultiUnit() 319–320
GeoEngKill() 321
GeoEngIsEval() 320
GeoEngMatchAddress() 322, 325
GeoEngMatchFormattedAddress() 325, 328
GeoEngMatchHwyExit() 328
GeoEngNAD27ToNAD83() 330
GeoEngNAD83ToNAD27() 331
GeoEngParseLastLine() 332
GeoEngSetCfgParms() 333–334
GeoEngSetInteractive() 335
GetCandidateAt() 164
GetCandidateAtV() 218
GetCandidateCensusBlockIDAt() 164
GetCandidateCensusBlockIDAtV() 219
GetCandidateCityAt() 165
GetCandidateCityAtV() 219

GeoEngine API function calls

GeoEngine API function calls

GeoEngine API
  list of function calls 278–279
GetCandidateFirmAt() 165
GetCandidateFirmAtV() 219
GetCandidateLatitudeAt() 166
GetCandidateLatitudeAtV() 220
GetCandidateLongitudeAt() 166
GetCandidateLongitudeAtV() 220
GetCandidatePlus4At() 167
GetCandidatePlus4AtV() 221
GetCandidatePrecisionAt() 167
GetCandidatePrecisionAtV() 221
GetCandidateResultCodeAt() 168
GetCandidateResultCodeAtV() 221
GetCandidateStateAt() 168
GetCandidateStateAtV() 222
GetCandidateStreetAt() 169
GetCandidateStreetAtV() 222
GetCandidateZIPAt() 169
GetCandidateZIPAtV() 223
GetFullName() 170
GetName() 170
GetVersion() 170

H

highway exit 82

I

input column settings
  saving to a table 50

Installation
  MapMarker APIs 26
  MapMarker Server 26
  MapX 4.0 users 27
  moving data and program files 30
  multiple versions of MapMarker 27
  procedure 18
  removing MapMarker 27
  shared 23
  unlocking DPV 25
  Windows registry 28

interactive geocoding 41, 43, 57, 65
intersections
  customized user dictionaries 110
  matching to 56

L

locate airport 81
locate highway exit 82
log file preferences 47
longitude/latitude columns 32–33, 35

M

map catalog
  creating 95
  remote mappable tables 94

MapMarker API
  creating standalone geocoding applications 266, 277, 385
  GeoEngine error codes 395, 403
  list of function calls 278
  MapBasic and Visual Basic notes 277

MapMarker API function calls list 278–279

MapMarker client/server toolkit
  MapMarker OCX 120

MapMarker geocoder control 120

MapMarker geocoder control (OCX)
  adding to client applications 121–122
  events and methods 125–126
  fall back to zip code centroids 125
  geocoding return values 124–125
  installing manually 116
  OLE control object 120
  setting properties 123

MapMarker OCX
  installing 26

MapMarker RPC Server
  shutting down 120
  starting 119

MapMarker server
  geocoding from remote workstations 116
  geocoding via MapMarker OCX 124–125
  installing 26
  installing manually 116
  request time outs 118
  running 117

MapMarker server RPC API
  list of function calls 231

MapMarker Streets
  using with MapMarker 14

MapX 4.x
  installation considerations 27

match settings
  specifying 53

matching accuracy
  strategies 53–54

matching addresses
  displaying match candidates on a map 373
  expanding search area 43
  input columns 32, 37, 39
  output columns 32–33, 35
  with geographic data 10
matching conditions
  CASS certification 50
  changing for interactive geocoding 43
  fallback settings 48
  multiple matches 49
  relaxing 56
  specifying 47–48
  trade-offs 52
matching trade-offs 52
metadata
  saving settings to a table 50
methods and events
  MapMarker OCX 125–126
mixed case street output 35
modify table structure 86
moving data 30

N
NAD83 and NAD27 datums
  address dictionary 376–377
  choosing 377
  ODBC tables 378
  preferences 372
networks
  installing or upgrading from 23
No data
  result codes 385
non-match codes
  result codes 65

O
ODBC tables
  datums 378
  supported data types 103
  transaction control 102
offset preferences 372
OLE automation API
  creating a custom control 130
  flow diagram 130
  list of ASP/VBScript methods 202–203
  list of ColdFusion methods 186–187
  list of OCX methods 135, 137
  programming example 174, 184
  programming example in ASP 224, 226
  summary of properties, events, methods 131

OLE automation methods
  ClearDialogText() 137
  Connect() 137–138
  Disconnect() 138
  DoGeocode() 139
  DoSetProperties() 139
  DpvGeocodeAddress() 139–140
  DpvGeocodeAddressLastLine() 141–142
  DpvGeocodeAddressLastLineWithSerial() 142–143
  DpvGeocodeAddressWithSerial() 143–144
  GeocodeAddress() 145, 147
  GeocodeAddressEx() 147–148
  GeocodeAddressLastLine() 148–149
  GeocodeAddressLastLineEx() 150
  GeocodeAddressLastLineWithSerial() 151
  GeocodeAddressWithSerial() 152–153
  GeocodeAddressWithSerialEx() 153
  GeocodeCheckDbAvailability() 155
  GeocodeFreeSet() 155
  GeocodeGetCandidates() 156–157
  GeocodeGetErrorText() 157
  GeocodeGetHwyExitCandidate() 158
  GeocodeGetServerVersion() 159
  GeocodeGetStatesFound() 159
  GeocodeGetStatesLicensed() 160
  GeocodeHwyExit() 160
  GeodecIsCandidateMultiUnit() 161
  GeocodePostalCentroid() 162–163
  GeocodePostalCentroidWithSerial() 163–164
  GetCandidateAt() 164
  GetCandidateCandidateMultiUnit() 164
  GetCandidateCityAt() 165
  GetCandidateCensusBlockIDAt() 165
  GetCandidateFirmAt() 165
  GetCandidateHwyExitCandidate() 166
  GetCandidateLatitudeAt() 166
  GetCandidateLongitudeAt() 167
  GetCandidatePlus4At() 167
  GetCandidatePrecisionAt() 167
  GetCandidateResultCodeAt() 168
  GetCandidateResultCodeAt() 168
  GetCandidateStateAt() 168
  GetCandidateStringAt() 169
  GetCandidateStreetAt() 169
  GetCandidateZIPAt() 169
  GetFullName() 170
  GetName() 170
  GetVersion() 170
  RefreshDialog() 170
  SelectCandidateAt() 171
OLE automation methods for ASP/VBScript
DpvGeocodeAddressLastLineV() 205–206
DpvGeocodeAddressV() 204
GeocodeAddressExV() 208
GeocodeAddressLastLineExV() 211
GeocodeAddressLastLineV() 210
GeocodeAddressV() 207
GeocodeFreeSetV() 212
GeocodeGetCandidatesV() 212
GeocodeGetErrorTextV() 213
GeocodeGetHwyExitCandidateV() 213
GeocodeGetServerVersionV() 215
GeocodeGetStatesFoundV() 215
GeocodeGetStatesLicensedV() 215
GeocodeHwyExitV() 216
GeocodeIsCandidateMultiUnitV() 217
GeocodePostalCentroidV() 217–218
GetCandidateAtV() 218
GetCandidateCensusBlockIDAtV() 219
GetCandidateCityAtV() 219
GetCandidateFirmAtV() 219
GetCandidateLatitudeAtV() 220
GetCandidateLongitudeAtV() 220
GetCandidatePlus4AtV() 221
GetCandidatePrecisionAtV() 221
GetCandidateResultCodeAtV() 221
GetCandidateStateAtV() 222
GetCandidateStreetAtV() 222
GetCandidateZIPAtV() 223

OLE automation methods for ColdFusion
GeocodeAddress_A() 187
GeocodeAddressEx_A() 188
GeocodeAddressLastLine_A() 189
GeocodeAddressLastLineEx_A() 190
GeocodeAddressWithSerial_A() 191, 193
GeocodeAddressWithSerialEx_A() 192
GeocodeFreeSet_A() 193
GeocodeGetErrorText_A() 193
GeocodeGetHwyExitCandidate_A() 194
GeocodeGetStatesFound_A() 195
GeocodeGetStatesLicensed_A() 196
GeocodeHwyExit_A() 196
GeocodeIsCandidateMultiUnit_A() 198
GeocodePostalCentroid_A() 198
GeocodePostalCentroidWithSerial_A() 199

OLE automation objects
CAddressList 171
output column settings
saving to a table 50
output columns
geocoding results 32–33, 35
output point
attribution type 85
output point attribution 85, 88

P
performance increasing 16, 384
positional accuracy
geced records 54
postal centroid match
result codes 65
preferences
CASS certification 50
datum 372
dictionary 371
fallback settings 48
gecoding 45, 47
log file 47
maps 373
match restrictions 47–48
multiple matches 49
offset 372
startup 35–36, 370
system 370, 374

Program files
list of 393
moving 30

Q
quick find
settings 80
quick find command
finding a single address 78–79
seeing your match 58
quick find settings 79–80
quick geocode button 78

R
RDBMS 94
RefreshDialog() 170
re-geocoding 45, 384
registry entries 405
related table geocoding 98, 101
relational database management systems 94
relative matching 52
relaxing match conditions 56
changing options settings 56
remote database geocoding
geocoding procedure 96, 98
preparation procedures 95
remote mappable tables 94, 96
rollback segment limit 103
supported data types 103
transaction control 102
remote mappable tables
   making a remote table mappable 96
   MapInfo map catalog 94
   setting datums 97–98
   supported data types 94
   supported spatial index types 94
removing MapMarker from your system 27
result code breakdown 65
result codes 65
   best match from multiple candidates 64
   no data 385
   non-match codes 65
   output columns 32–33, 35
   postal centroid matches 65
   single close match 63
   unmatched rows 384
results tables
   related table geocoding 98
rollback segment limit
   remote databases 103
RPC API function call
   GeocodeHwyExit() 262
RPC API function calls
   DpvGeocodeAddress() 232–233
   DpvGeocodeAddressLastLine() 234–235
   DpvGeocodeAddressLastLineWithSerial() 235, 237
   DpvGeocodeAddressWithSerial() 237–238
   GeocodeAddress() 239–242
   GeocodeAddressLastLine() 235, 242–243
   GeocodeAddressLastLineEx() 243–244
   GeocodeAddressLastLineWithSerial() 245–246
   GeocodeAddressLastLineWithSerialEx() 246–247
   GeocodeAddressWithSerial() 248–249
   GeocodeAddressWithSerialEx() 249–250
   GeocodeCheckDbAvailability() 251
   GeocodeCheckDbSet() 251
   GeocodeGetCandidate() 252–253
   GeocodeGetCandidateCoords() 255
   GeocodeGetCandidateEx() 253–254
   GeocodeGetCandidateWithDpv() 255–256, 258
   GeocodeGetErrorText() 258
   GeocodeGetHwyExitCandidate() 259
   GeocodeGetServerVersion() 260
   GeocodeGetStatesFound() 260
   GeocodeGetStatesLicensed() 260
   GeocodeHwyExit() 261
   GeocodeIsCandidateMultiUnit() 262
   GeocodePostalCentroid() 263
   GeocodePostalCentroidWithSerial() 264
   GeocodeShutdown() 264
running MapMarker 37

S
   sample attribution file 91
   saving settings to a table 91
   searching
      quick find command 78–79
   select attribute tables 87
   select input columns 32
   SelectCandidateAt() 171
   single close match
      result codes 63
Solaris
   RPC Server 119–120
   spatial index type 94
   spatial index types
      remote mappable tables 94
standalone geocoding applications
   creating with MapMarker API 266, 277, 385
startup preferences
   setting 35–36, 370
stop attribution 90
street intersections
   matching to 56
street names
   relaxing match conditions 56
street suffix abbreviations 395, 415
StreetWorks
   using with FindAddress 389
system preferences
   datum 372
   dictionary 371
   maps 373
   offset 372
   setting 370
   startup 370, 374

T
   tabbedAddress field
      CANDIDATE_ADDRESS structure 350
   table attribution 85–88
      column value 85, 89
      output point 85, 88
      preparing your table 86–88
      selecting a table 87
Tables
   opening in MapInfo 384
   tables
      editing records while geocoding 44
      ODBC 102, 378
      saving settings to 50
      sorting before geocoding 32
transaction control 102
U
uninstalling  
   removing MapMarker from your system 27
unlocking DPV 71–72
unmatched rows  
   result codes 384
updated records  
   re-geocoding a table 45, 384
upgrading to current version 17
user dictionaries  
   considerations 109–110
   creating 106, 109
   FindAddress utility 389
   intersections 110
   system preferences 371
utilities  
   append.mbx 389, 391
   FindAddress 388–389

W
Windows console application, running MapMarker server 118
Windows registry  
   installation considerations 28
Windows service, running MapMarker server 117–118

Z
ZIP Codes  
   matching to centroids 57, 77
   ranged by state 411
   relaxing match conditions 56