Table of Contents

Chapter 1: Introduction .......................................................... 7
  What Is Geocoding? ......................................................... 8
  What Is MapMarker? ....................................................... 8
  MapMarker Developer Product ........................................... 8
    MapMarker Server and Developer Tools ............................. 8
    MapMarker Desktop Application ................................. 9
  MapMarker Standard or MapMarker Plus ............................ 9
  What’s New in MapMarker Plus 11.0? ................................. 10
  MapMarker Documentation Set ........................................ 10
    MapMarker Developer Guide ....................................... 11
    MapMarker Desktop User Guide ................................... 11
    MapMarker Javadocs ............................................... 11
    MapMarker Installation Guide ................................... 12
    MapMarker Release Notes ....................................... 12
    Online Help .......................................................... 12
  MapMarker Streets ...................................................... 12

Chapter 2: Installation .......................................................... 13
  Installation Overview .................................................. 14
    Making the Transition from MapMarker Plus 10.x to MapMarker Plus 11.0 .................................. 14
    Address Dictionary ................................................ 14
    DPV™ (Delivery Point Validation) Data .......................... 14
  System Requirements .................................................. 15
    Operating Systems .................................................. 15
    Web Servers ......................................................... 16
    Minimum Requirements ........................................... 16
    Recommended Requirements .................................... 16
    High Performance Requirements ................................ 16
  System Recommendations ............................................ 16
    Storing Data on Your Hard Drive ................................ 16
    Performance Tuning ............................................... 17
  Windows Installation Procedure .................................... 17
  UNIX and Linux Installation Procedure ............................ 23
    Installing in Console Mode ....................................... 23
  Re-Installing MapMarker ............................................... 24
  MDAC 2.7 and MapMarker ODBC Support ............................ 25
**MapMarker Plus 11.0**

© 2005 MapInfo Corporation. All rights reserved.

---

**Developer Guide Table of Contents**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting MapMarker</td>
<td>26</td>
</tr>
<tr>
<td>Desktop</td>
<td>26</td>
</tr>
<tr>
<td>Developer</td>
<td>26</td>
</tr>
<tr>
<td>JNI Adapter Configuration File</td>
<td>26</td>
</tr>
<tr>
<td><strong>Chapter 3: Migrating to MapMarker Plus 11.0</strong></td>
<td>27</td>
</tr>
<tr>
<td>MapMarker Geocoding API Development</td>
<td>28</td>
</tr>
<tr>
<td>Java Developers</td>
<td>28</td>
</tr>
<tr>
<td>J Server 2.x API Status</td>
<td>28</td>
</tr>
<tr>
<td>J Server 3.x API Status</td>
<td>28</td>
</tr>
<tr>
<td>Windows Developers</td>
<td>29</td>
</tr>
<tr>
<td>Migration Path for Windows Developers</td>
<td>30</td>
</tr>
<tr>
<td>Solaris, HP-UX, and Linux Developers</td>
<td>30</td>
</tr>
<tr>
<td>Migration Path for Solaris, HP-UX, and Linux Developers</td>
<td>30</td>
</tr>
<tr>
<td>Database Developers</td>
<td>31</td>
</tr>
<tr>
<td>Migration Path for Database Developers</td>
<td>31</td>
</tr>
<tr>
<td><strong>Making the Transition to MapMarker Plus 11.0.</strong></td>
<td>31</td>
</tr>
<tr>
<td>MapMarker Plus 10.x</td>
<td>31</td>
</tr>
<tr>
<td>JNI Adapter</td>
<td>32</td>
</tr>
<tr>
<td><strong>Compiling and Running Existing Applications.</strong></td>
<td>32</td>
</tr>
<tr>
<td>Windows Applications</td>
<td>32</td>
</tr>
<tr>
<td>UNIX Applications</td>
<td>33</td>
</tr>
<tr>
<td>Other Configuration File Settings</td>
<td>34</td>
</tr>
<tr>
<td><strong>Chapter 4: Designing Geocoding Applications</strong></td>
<td>35</td>
</tr>
<tr>
<td>Choosing a Development Tool</td>
<td>36</td>
</tr>
<tr>
<td>MapInfo’s Geocoding Interoperability</td>
<td>36</td>
</tr>
<tr>
<td>Geocoding Terms</td>
<td>38</td>
</tr>
<tr>
<td>Defining the Geocoding Model</td>
<td>39</td>
</tr>
<tr>
<td>Geocoding Preferences</td>
<td>39</td>
</tr>
<tr>
<td>Use of Geocoded Data</td>
<td>40</td>
</tr>
<tr>
<td>Executing the Application</td>
<td>40</td>
</tr>
<tr>
<td><strong>Geocoding Application Design Overview</strong></td>
<td>40</td>
</tr>
<tr>
<td>Types of Geocoding</td>
<td>41</td>
</tr>
<tr>
<td>Building an Input Address</td>
<td>42</td>
</tr>
<tr>
<td>Using Preferences</td>
<td>43</td>
</tr>
<tr>
<td>Sending a Request</td>
<td>45</td>
</tr>
<tr>
<td>Verifying a Response</td>
<td>45</td>
</tr>
<tr>
<td>Using Geocode Result Information</td>
<td>45</td>
</tr>
</tbody>
</table>
# MapMarker Plus 11.0 Developer Guide Table of Contents

## Chapter 5: Using the MapMarker Plus Java API ........................................ 47

- **Overview** .................................................. 48
  - Using the MapMarker Plus Java USA API ........................................ 48
  - Using the MapMarker Generic Java API ........................................ 48
- **Developing Geocoding Applications in Java** .................................... 48
  - Creating a Geocoding Application in Java ....................................... 48
  - Creating a Web Geocoding Client in Java ....................................... 54
  - Sample Code ........................................................................... 55
- **Browsing Addresses** ........................................................................ 58
- **Geocoding to City Centroids** .......................................................... 59
- **Geocoding to Highway Exits** .......................................................... 61
  - Finding Airports by State ......................................................... 62
  - Finding Airports by Code .................................................................. 63

## Chapter 6: MapMarker Sample Application ........................................... 64

- **Requirements and Setup** .............................................................. 65
- **Using the Sample Application** ....................................................... 65
  - Testing the Server ....................................................................... 66
  - Setting Preferences ....................................................................... 66
  - Setting Must Match Options .......................................................... 66
  - Postal Geocoding .......................................................................... 66
  - Area Geocoding ............................................................................ 66
  - Address Geocoding ....................................................................... 67
  - Viewing Candidates ....................................................................... 67
  - Candidate Statistics ...................................................................... 67

## Chapter 7: Deploying Applications ....................................................... 68

- **Deploying MapMarker Plus as a Servlet** ......................................... 69
  - Using the Default Tomcat Servlet Container ...................................... 69
  - Deploying MapMarker Server in Other Web Environments ................ 69
- **Integration with MapMarker for Other Countries** .............................. 69

## Chapter 8: Creating and Using Custom Address Dictionaries .................. 71

- **Creating a Custom Dictionary** ....................................................... 72
  - Required and Optional Fields ......................................................... 72
  - User Dictionary File Format ............................................................ 73
  - Data Dictionary Interface .................................................................. 74
  - **Using Custom Dictionaries** ......................................................... 76
  - Configuring User Dictionaries on the Server ...................................... 76
  - User Dictionary Preferences ............................................................. 76
  - Client Preferences .......................................................................... 78
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? .................................................. 8
- What Is MapMarker? ................................................. 8
- MapMarker Standard or MapMarker Plus .................. 9
- What’s New in MapMarker Plus 11.0? ...................... 10
- MapMarker Documentation Set ................................. 10
- MapMarker Streets .................................................. 12
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 Developer Product

The MapMarker Developer product includes the MapMarker server and developer tools, as well as the MapMarker desktop product.

MapMarker Server and Developer Tools

The MapMarker Plus developer product is a server and a developer tool that allows you to add geocoding functionality to your desktop or Web application.
The MapMarker server enables multiple simultaneous geocoding requests to be served from a single MapMarker engine. Using the Java API, developers can write client/server or standalone geocoding applications. An OCX is also included 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 Java or C, the complete geocoding engine API is provided as well.

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™ (Coding Accuracy Support System) requirements using MapMarker Plus Address Dictionary.
- Use DPV™ (Delivery Point Validation) while geocoding in CASS mode to verify whether a geocoded address exists.
- Geocode remote tables via ODBC.

MapMarker Standard or MapMarker Plus

The MapMarker desktop product is available in two versions: Standard or Plus. If you purchased MapMarker Standard, you receive the standard Address Dictionary. MapMarker Standard is released twice a year.

If you purchased the MapMarker 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.
What’s New in MapMarker Plus 11.0?

- **Java engine**—The biggest change in MapMarker Plus is that the product is now built on a Java-based geocoding engine. You can develop standalone or client/server applications using the Java API and Java server. All the same features that were available in previous versions are available in MapMarker Plus 11.0, including geocoding to locations such as airports and highway exits. The user interface of the desktop product is the same as in previous versions of MapMarker.

- **LACSLink™**—The LACSLink technology from the USPS® has been incorporated into MapMarker Plus for use in CASS geocoding. LACSLink provides the new address for an input address that has been converted to a street-style address for E911 purposes, or for other USPS changes.

- **User Dictionaries**—The creation of user dictionaries has been enhanced so that you can now use multiple user dictionaries at once when you geocode. The limit of five user dictionaries in previous versions of MapMarker has been removed.

  Note that user dictionaries you created in an earlier version of the product are not compatible with MapMarker 11.0 because of the move to the Java engine. You will need to re-create them in MapMarker 11.0. If this is not possible, a conversion tool enables you to convert a dictionary created in a previous version to a MapMarker Plus 11.0 user dictionary. We recommend, however, that if you have the original source data that you simply re-create the dictionary using the User Dictionary Wizard in the MapMarker desktop product.

  Using the Java API, you can set dictionary preferences and specify the path to each dictionary.

- **To make the transition to Java easier, we have provided a JNI Adapter which acts as a translator for the C engine GeoEngine API calls. Any existing applications written in C can be run in MapMarker Plus 11.0 using the adapter.**

- **U.S. city (geographic centroid) geocoding**—Geocode to U.S. city centroids using the Java API. Note this feature is not available in the desktop product.

MapMarker Documentation Set

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

- MapMarker Desktop User Guide in print and in PDF format
- MapMarker Developer Guide in PDF format
- MapMarker Installation Guide in PDF format
- Online Help for both the MapMarker product and installer
MapMarker Developer Guide

The MapMarker Developer Guide explains how to use the MapMarker Java API to write geocoding applications and how to use your existing MapMarker C applications with it. It provides information on:

- Using existing C applications with the MapMarker Java API using the JNI adapter
- Migrating existing applications written with the MapMarker C APIs or the MapMarker J Server to the MapMarker Java API.
- Designing geocoding applications.
- Writing applications using the MapMarker Java API that enable users to geocode to streets, city centroids, airports, and highway exits.
- Creating and using user dictionaries.

MapMarker Desktop User Guide

This book is designed to help you use MapMarker to the fullest. It 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 DPV; finding airports, highway exits, and single addresses; batch geocoding, and table attribution. The Desktop User Guide also provides a number appendixes that provide reference information for the MapMarker desktop product. It includes:

- 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.
- Frequently Asked Questions—A list of frequently asked questions.
- MapMarker Program Files—A list of the files that are installed with MapMarker.
- Street Suffix Abbreviations—A list of USPS standard abbreviations for words that frequently appear in street addresses.

In addition, the MapMarker User Interface Reference is available as an additional appendix in the PDF version of the User Guide. It is a reference of all the commands and dialog boxes in MapMarker.

MapMarker Javadocs

The MapMarker javadocs provide documentation on the Java API. To access the javadocs, browse to http://localhost:8095 on a computer where the MapMarker Server is running and click on "MapMarker Plus USA Java API" in the box labeled MapMarker Documentation.
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


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. Refer to this chapter for step-by-step instructions and other issues related to installation.

In this chapter:

- Installation Overview .................................................. 14
- System Requirements .................................................... 15
- System Recommendations ............................................. 16
- Windows Installation Procedure ................................. 17
- UNIX and Linux Installation Procedure ...................... 23
- Re-Installing MapMarker ............................................. 24
- MDAC 2.7 and MapMarker ODBC Support ................. 25
- Starting MapMarker ................................................... 26
- JNI Adapter Configuration File ................................. 26
Installation Overview

The MapMarker Plus developer product comes on one DVD-ROM and includes the desktop software, the Java SDK, and data. The installer installs all the product components in one easy procedure. The installer also handles all registry keys. The components installed during a complete installation are:

- MapMarker Plus desktop software (Windows only)
- MapMarker Plus Address Dictionary
- Java SDK
- Additional Developer Tools
- Servlet Container

Making the Transition from MapMarker Plus 10.x to MapMarker Plus 11.0

MapMarker Plus 11.0 marks a departure from previous version of the MapMarker product. Version 11.0 is built on the Java platform. Both versions of the product can operate side-by-side on the same computer. The only difference that you will see in the way that MapMarker Plus 11.0 developer product looks and performs is in the installation.

If you are a MapMarker Plus 10.x customer updating to MapMarker Plus 11.0, you will not be able to perform an upgrade to install it. You will need to do a full installation of the product. You may also notice that the installation procedure is slightly different than in previous versions. We have made every effort to make installing the product as easy as possible.

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, or ZIP + 4 centroids.

During the installation, you will be asked to supply a data licensing code for the data that you purchased. To obtain the code, call the MapMarker License Authorization Desk at (800) 552-2511 option 3, between 8 a.m. and 7 p.m. ET, Monday through Friday. When you enter your code, the installer places a license on your system that matches your purchase and entitles you to access the appropriate data. The license file 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 Choose Install Set dialog, de-select any states that you do not want to install.

DPV™ (Delivery Point Validation) Data

The licensing code also handles the installation of the DPV data, as well as makes it usable. In previous versions of MapMarker Plus, an additional “unlocking” step was necessary to make the DPV data usable. This unlocking step is now only necessary to re-enable DPV in the event of a usage violation.

If you purchased DPV, the license code is valid for fourteen days from the date that you obtained the code. If you use a license code that has expired, you will receive the following message:
You will be able to continue with the installation of the MapMarker software, developer components, and Address Dictionary data, but to enable DPV you will need to obtain a new authorization code from Customer Service, and then use the CASS Authorization Utility to enable DPV use.

To add DPV to an existing installation, you will need to remove your MapMarker installation and install it again. Please see Re-Installing MapMarker on page 24 for more information.

System Requirements

This section describes the software and hardware requirements for running MapMarker Plus.

Operating Systems

MapMarker Plus runs on the following operating systems:

- Windows 2000 with SP4
- Windows XP with SP2
- Windows 2003 Enterprise Edition
- Windows 2000 Advanced Server with SP4
- Solaris 2.9, 2.10
- HP-UX 11.0
- Red Hat Linux

Java Virtual Machine

MapMarker Plus requires a Java Virtual Machine (JVM) to run. The following JVMs are supported:

- JVM v. 1.4.1 or later

MapMarker ships with JVM v. 1.5.

During the installation, you will have the option of installing one of the JVMs that ships with MapMarker, or using one that you already have installed on your computer.
Web Servers

The following Web servers are supported:

- Tomcat 4.1.1.8, 5.x (MapMarker ships with v. 5.0)
- Sun One 7.0
- BEA WebLogic 8.1
- WebSphere 5.1 and 6.0

Minimum Requirements

The minimum system requirements for MapMarker Plus 11.0 on a full installation are:

- 800 Mhz Pentium® processor or equivalent
- 256 MB RAM
- 5 GB available disk space

Recommended Requirements

The recommended system requirements on a full are:

- 1 Ghz processor or better
- 512 MB RAM
- 10 GB available disk space

High Performance Requirements

For high performance, the following system requirements should be in place:

- 2.5 Ghz dual processor or better
- 1 GB RAM or better
- 10 GB available disk space

The MapMarker software requires about 25 MB of available hard drive space. The Plus Address Dictionary requires approximately 2.63 GB (for nationwide data) of disk storage space. The DPV data requires approximately .5 GB 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 local hard drive. The data resides in the data directory under the MapMarker program directory (default is c:\Program Files\MapInfo\MapMarker_USA\data).
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.

Windows Installation Procedure

1. Place the MapMarker DVD into your DVD-ROM drive.
   On Windows platforms, if the installation does not automatically start, choose Run from the Windows 2000/2003/XP START button. From the Run dialog, type D:\SETUP.EXE in the Open command box, where D is the drive letter of your DVD-ROM. Click OK. The MapMarker Installation Options screen displays on your screen.

2. Click the INSTALL MAPMARKER... button.

3. The Introduction dialog displays on the screen.

4. Click NEXT to continue. The License Agreement dialog displays on the screen.
5. After reading the License Agreement, click the **I ACCEPT THE TERMS OF THE LICENSE AGREEMENT BUTTON**, and then click **NEXT** to continue. The Enter Serial Number dialog displays on the screen.

6. At the Enter Serial Number dialog, type your MapMarker Plus serial number. The serial number is located on a sheet of stickers in the MapMarker Plus package. Click **NEXT** to continue.

The Enter Unlocking Code dialog displays on the screen.
7. At the Enter Unlocking 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 have purchased a single state, or a six-state pack, you will be able to install those states after entering your unlocking code.

8. Click **Next** to continue. The Choose Install Set dialog displays on the screen.

9. In the Typical Install Set, all of the product components are installed by default. To see a description of each component, click on the component to highlight it. A description for the component displays in the Description box.

To select which components you want to install, choose Custom in the Install Set drop-down list. Clear the check boxes of the components you do not want to install.

10. When you are finished, click **Next** to continue. The Installation Type dialog displays on the screen.
Select the location of the MapMarker Plus software. You can use the default location: 

C:\Program Files\MapInfo\MapMarker\MapMarker_USA

or click **CHOOSE** to display the Browse for Folder dialog, where you can navigate to the desired location. Click **OK** in the Browse for Folder dialog when you are finished.

If you decide after selecting an alternative location that you would prefer the default location, 
click **RESTORE DEFAULT FOLDER** in the Choose Install Set dialog.

11. Click **NEXT** in the Choose Software Install Folder dialog to continue. The Choose a Data Folder 
dialog displays on the screen.
12. Select the location of the MapMarker data. You can use the default location:
   C:\Program Files\MapInfo\MapMarker\MapMarker_USA\Data
   or click CHOOSE to display the Browse for Folder dialog, where you can navigate to the desired
   location. Click OK in the Browse for Folder dialog when you are finished.
   If you decide after selecting an alternative location that you would prefer the default location,
   click RESTORE DEFAULT FOLDER in the Choose a Data Folder dialog.

13. Click NEXT in the Choose a Data Folder dialog to continue. The Choose Java Virtual Machine
   dialog displays on the screen.

   Do one of the following:
   - Select the Install a Java VM specifically for this application button to install the Java virtual
     machine that comes with MapMarker Plus.
   - Select the Choose a Java VM already installed on this system button to select a Java virtual
     machine that you already have installed on the computer.
Select the Search for Others button to have the installer search your computer for an installed JVM. Select the Choose Another button if you know where your JVM is installed. You must have a JVM v. 1.4.1 or later.

**Note:** If you select the Choose a Java VM already installed on this system button and there is no VM installed on your computer, you will receive an error message when you click **Next**.

14. Click **Next** to continue. The Sample Web Environment parameters dialog displays. Enter the hostname and port you want to use with the sample web environment. The default hostname is the name of your computer, and the default port number is 8095.

15. Click **Next** to continue. The Choose Shortcut Folder dialog displays on the screen. Click the corresponding button of the desired location of the product icons. Click the Don’t create icons button if you do not want to create product icons.

If you clicked the **Other** button, and you want to use a location different from the default, click the **Choose** button to navigate to the desired location.

16. Click **Next** to continue. The Pre-Installation Summary dialog displays on the screen.
17. Review your install selections. To edit any of the selections, click the Previous button to go back to the earlier dialogs.

18. When you are satisfied with your installation choices, click **INSTALL**. A progress dialog displays as the software and data is installing.

19. When the installation is finished, the Install Complete dialog displays on the screen. Click **DONE** to exit the installer.

**UNIX and Linux Installation Procedure**

On HP-UX, Solaris, or Linux platforms, do the following:

1. Place the MapMarker DVD into your DVD-ROM drive.
2. Open a terminal window.
3. At the command prompt enter: `sh <path to media>/startinstall.sh` to start the MapMarker installer, where “path to media” is the full path to the mounted installer media, e.g., `/dvdrom/dvdrom0`.

**Installing in Console Mode**

To run the installation program in console mode:

1. Change directories to the instdata directory on the installation media, and then to the directory appropriate to your platform (HP-UX, Solaris, or Linux).
2. In your platform’s directory, descend to the directory named vm.
3. Enter `install.bin -i console` or `sh /install.bin` as appropriate for your platform.

The installer detects that there is no user interface and runs itself in console mode.

**Note:** The MapMarker console mode installation works only with the DVD media. If you requested CDs, you will need to perform the GUI installation.
Re-Installing MapMarker

If you purchase components and data subsequent to your MapMarker purchase, such as DPV or licenses for additional states, you will need to remove your current installation of MapMarker Plus, and then install it again.

If you re-launch the installation program from the DVD, it will guide you through the removal and re-installation process:

1. Place the MapMarker DVD into your DVD-ROM drive.
   If the installation does not automatically start, from the Windows 2000/2003/XP START button, choose RUN. From the Run dialog, type D:\SETUP.EXE in the Open command box, where D is the drive letter of your DVD-ROM. Click OK.

2. A dialog displays on the screen with a message, indicating that you already have MapMarker Plus installed and that to make change to your current installation, you must remove MapMarker and reinstall it.

3. Click REMOVE to remove your installation of MapMarker Plus. When the existing installation of MapMarker Plus has been removed, the installer continues, and the Enter Serial Number dialog displays on the screen. Your serial number is already filled in.

4. Click NEXT to continue. The Enter Unlocking Code dialog displays on the screen. The license code from your original installation is already filled in. You will need to call Customer Service for a new license code in the following circumstances:
   • You are a current DPV customer and your original license code is over fourteen days old.
   • You are installing DPV as an add-on.
   • You are installing additional data.
Please call MapInfo Customer Service: (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 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 proceed to the next dialog without changing your unlock code, and a new code was needed, you will need to exit and restart the installer to change the code.

5. Click **Next** to continue. The rest of the installation procedure is the same as it is for a new install. For more information, please see **Windows Installation Procedure on page 17**.

**MDAC 2.7 and MapMarker ODBC Support**

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 11.0 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
- Microsoft Access (*.mdb) v. 4.00

MapMarker also supports the following Oracle driver:

- Oracle 9i Release 2

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

Please follow these instructions to start the MapMarker Plus desktop and developer products.

Desktop

On Windows 2000/2003/XP, choose MapMarker Plus 11.0 Desktop from the Start menu or Shortcut menu.

Developer

To start the Java server On Windows 2000/2003/XP, choose MapMarker USA> Start MapMarker Plus USA Server from the Start menu or Shortcut menu.

You can also start the Java server by running startup.bat (Windows) or startup.sh (Solaris, HP-UX, and Linux). This file is located in the \SDK\MapMarker-4.0\bin folder.

   Note: On HP-UX and Solaris, you can start either the RPC or Java server. On Linux, only the Java server is available.

JNI Adapter Configuration File

The adapter uses a configuration file that is put down at the time of installation, geojni.cfg. On Windows systems it is located in the MapInfo\MapMarker\MapMarkerPlusUSA\SDK\Additional Tools folder where MapMarker Plus is installed.

On UNIX systems it can be located anywhere and have any name. The environment variable CONFIG_FILE_PATH must be set to the full path of the file (including the filename) and exported in the environment of an application using the adapter.

   export CONFIG_FILE_PATH=/opt/mapinfo/mapmarker/geojni.cfg

You may want to modify this file to increase the memory usage. You also may want to take note of the location of the file, for technical support purposes.

For more specific information about the configuration file settings, see JNI Adapter Configuration File Settings in Appendix B on page 83.
Migrating to MapMarker Plus 11.0

This chapter describes the support available for developers who want to use their C-based geocoding applications in MapMarker 11.0, and discusses when and how existing applications should be migrated to the Java API.

In this chapter:

- MapMarker Geocoding API Development ....................... 28
- Java Developers ...................................................... 28
- Windows Developers .............................................. 29
- Solaris, HP-UX, and Linux Developers ....................... 30
- Database Developers ............................................. 31
- Making the Transition to MapMarker Plus 11.0 ........... 31
- Compiling and Running Existing Applications .............. 32
MapMarker Geocoding API Development

MapInfo has offered a variety of geocoding tools and APIs to Windows and UNIX developers writing in the C and Java languages. MapMarker has traditionally had a number of C APIs for Windows developers, a smaller subset of the same APIs for UNIX developers, and a Java solution integrating MapMarker J Server with MapMarker Plus for Java developers.

MapMarker Plus 11.0 is built on a Java geocoding engine and offers a full Java API that is supported on Windows, UNIX, and Linux platforms. With this release, the older APIs have been deprecated and the tools that MapMarker Plus 11.0 replaces have been retired.

This chapter summarizes what has been available for each platform up until the release of MapMarker Plus 11.0, the retirement or deprecation status of the tool or API, and the migration path available to developers on each platform. It also provides some information on making the transition to MapMarker Plus 11.0, and compiling and running existing applications.

Java Developers

Java developers currently use the MapMarker J Server API to create geocoding applications in Java. The MapMarker J Server is a servlet that uses JNI to access the C-based geocoding engine. J Server clients send requests via HTTP.

J Server 2.x API Status

This API is no longer supported.

Migration Path for Java Developers

Existing customers can still use this unsupported API with the MapMarker Plus 10.x product, but they must migrate to the MapMarker Plus 11.0 Java API or Envinsa Java API before the 10.x product is retired.

Although customers who migrate their applications from MapMarker J Server 2.x USA API to the MapMarker Plus 11.0 Java API will need to make moderate code changes, they will also gain some advantages. For example, customers that migrate to the MapMarker Plus 11.0 Java API will be able to:

- Utilize a pure Java solution and receive the latest features and software enhancements.
- Communicate directly to the Java geocoding engine or server.

J Server 3.x API Status

This API will be deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.
Migration Path for Java Developers

Existing customers can still use this API with the MapMarker Plus 10.x product, but they must migrate to the MapMarker Plus 11.0 Java API or Envinsa Java API before the 10.x product is retired.

Although customers who migrate their applications from MapMarker J Server 3.x USA API to the MapMarker Plus 11.0 Java API will need to make minor code changes, they will also gain some advantages. For example, customers that migrate to the MapMarker Plus 11.0 Java API will be able to:

• Utilize a pure Java solution and receive the latest features and software enhancements.
• Communicate directly to the Java geocoding engine or server.
• Ability to geocode to geographic centroids.

Windows Developers

In the C-based version of MapMarker Plus, Windows developers made use of a number of APIs available on Windows platforms. The following table lists each of the APIs, gives a brief description, and its deprecation status.

<table>
<thead>
<tr>
<th>Windows API</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE Automation API</td>
<td>The OLE Automation API is used to create customized OCX geocoding applications.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
<tr>
<td>ASP API</td>
<td>The ASP API is for developers who want to create geocoding applications that use ASP/VBScript.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
<tr>
<td>Cold Fusion API</td>
<td>This API is used for developing geocoding application with ColdFusion.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
<tr>
<td>RPC Server API</td>
<td>The MapMarker RPC API is for the C developer who wants to create geocoding applications that call the MapMarker RPC Server.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
<tr>
<td>GeoEng API</td>
<td>This is the C API for the MapMarker geocoding engine used for creating batch geocoding applications like MapMarker. The MapMarker geocoding engine and its API are packaged as a 32-bit Dynamic Link Library.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
<tr>
<td>MapBasic ‘geocoding’ API</td>
<td>In addition to the C interfaces, we have provided interfaces to the MapMarker API for MapBasic.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
<tr>
<td>VB 6.0 API</td>
<td>In addition to the C interfaces, we have provided interfaces to the MapMarker API for Visual Basic 6.0.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
</tbody>
</table>
Migration Path for Windows Developers

Developers can use their existing applications with MapMarker Plus 10.x or MapMarker Plus 11.x, either with the v. 10.x data refresh releases for those who want to stay with v. 10.x for the time being, or with the JNI adapter provided with MapMarker Plus 11.0 that enables existing C applications to be run with the MapMarker Plus 11.0 Java API.

Solaris, HP-UX, and Linux Developers

Non-Windows developers have had more limited choices on which API to use. Developers who write applications for the Solaris platform used the same GeoEng and RPC Server APIs that Windows developers used. The GeoEng API was also available for HP. There was no Linux support. The following table provides a description of each API and its status.

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoEng API</td>
<td>This is the C API for the MapMarker geocoding engine used for creating batch geocoding applications like MapMarker. The MapMarker geocoding engine and its API are packaged as a shared library or shared object.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
<tr>
<td>RPC Server API</td>
<td>The MapMarker RPC API is for the C developer who wants to create geocoding applications that call MapMarker RPC (Noble Net) Server.</td>
<td>Deprecated with the release of MapMarker Plus 11.0. It will no longer be supported with the release of MapMarker Plus 12.0.</td>
</tr>
</tbody>
</table>

Migration Path for Solaris, HP-UX, and Linux Developers

Customers can still use their existing applications with the MapMarker Plus 10.x or MapMarker Plus 11.x products. MapMarker Plus 10.x can be run with the v. 10.x data refresh releases for customers who want to continue with 10.x for a period of time before moving to MapMarker Plus 11.0. In MapMarker Plus 11.0, the JNI adapter enables existing C applications to be run with the MapMarker Plus Java API.

Developers who want Linux support must use MapMarker Plus 11.0.

Because the C APIs are deprecated, developers are encourage to migrate to the MapMarker Plus 11.0 Java API or Envinsa Java API.
Database Developers

MapInfo has offered three different products to database developers who want to include geocoding in their applications. The table below describes each one and gives its status.

<table>
<thead>
<tr>
<th>Product</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informix DataBlades</td>
<td>The Informix DataBlade product is a thin, pass-through layer on top of the MapMarker RPC Server, which provides an SQL interface for address geocoding.</td>
<td>Support for this product has been retired.</td>
</tr>
<tr>
<td>Oracle Geocoding Cartridge</td>
<td>The Oracle Geocoding Cartridge product is a thin, pass-through layer on top of the J Server client, which provide an SQL interface for address geocoding.</td>
<td>A software update is expected sometime after the MapMarker Plus 11.0 product ships. This update will add MapMarker Plus 11.0 functionality to the Oracle Geocoding Cartridge.</td>
</tr>
<tr>
<td>ESP SQL Server</td>
<td>The ESP SQL Server product is a thin, pass-through layer on top of the MapMarker RPC Server, which provides an SQL interface for address geocoding.</td>
<td>A software update is expected sometime after the MapMarker Plus 11.0 product ships. This update will add MapMarker Plus 11.0 functionality to the ESP SQL Server.</td>
</tr>
</tbody>
</table>

Migration Path for Database Developers

Customers who currently use the Oracle Geocoding Cartridge should use MapMarker J Server 3.1 with the MapMarker Plus 10.x product.

Customers who currently use ESP SQL Server should use the Windows RPC Server with the MapMarker Plus 10.x product.

Both products will be updated following the release of MapMarker Plus 11.0.

Making the Transition to MapMarker Plus 11.0

To ease the transition to MapMarker Plus 11.0, we have provided support for developers who wish to delay using MapMarker 11.0, or who want to use version 11.0 with their existing applications.

MapMarker Plus 10.x

The MapMarker Plus 10.x C-engine product will be supported during the v. 11.x release cycle, with quarterly data refreshes. These data refresh releases are data updates only. No updates will be made to the 10.x software.

Depending on your needs and the tools you are currently using, you may choose to migrate existing applications to MapMarker Plus 11.0 now, or wait until sometime before the 10.x series is retired to do so.
JNI Adapter

If you want to use your C-based geocoding applications with MapMarker 11.0, we have provided a JNI adapter that will enable you to run them.

The v. 10.x data refreshes and the JNI Adapter offer a temporary means of running your applications until you can migrate them to MapMarker Plus 11.0. The C APIs are deprecated and will no longer be supported with the release of MapMarker Plus 12.0. We strongly encourage developers to migrate existing applications to the MapMarker Plus 11.0 Java API, MapXtreme .NET or Envinsa .NET/Java API as soon as is practical.

Developers writing new applications should use the MapMarker Plus 11.0 Java API, MapXtreme .NET, or Envinsa .NET/Java API.

Compiling and Running Existing Applications

Geocoding applications written in C can be run using a JNI adapter that communicates with the MapMarker Java engine. The JNI adapter acts as a translator for the MapMarker 10.x version of the GeoEngine API, and enables your existing applications to be run in MapMarker Plus 11.0. This section explains how to compile and run Windows and UNIX applications.

Windows Applications

The following files must be included in your application. These files are located in the SDK\AdditionalTools\geoeng\include folder where MapMarker Plus is installed.

```
#include <geo.h>
#include <geoerror.h>
#include <geostd.h>
```

Link your application to mm32v11.lib. This file is located in the SDK\AdditionalTools\geoeng\lib folder where MapMarker Plus is installed.

The adapter uses a configuration file, geojni.cfg, plus backward compatibility libraries. You must copy these files to your application directory in order for it to run. These files are located in the root MapMarker installation directory.

- geojni.cfg
- mm32v11.dll
- mm32v10.dll
- mm32v9.dll
- mm32v8.dll
- mm32v7.dll
- mm32v6.dll
- mm32v5.dll

Ensure that the classpath in geojni.cfg is set to the location of the following files, which are located in the root MapMarker installation directory.
- MMUSALicenseProvider.config
- MMUSALicenseFactory.config
- USA_DataManagerSettings.properties

Make sure the classpath in geojni.cfg is set to the following jar files, which are located in the root MapMarker installation directory.

- micsys.jar
- miutil.jar
- mmj.jar
- mmjclient.jar
- mmjjni_usa.jar
- mmjclient_usa.jar
- mmj_usa.jar
- miscp.jar

**UNIX Applications**

For UNIX applications, the adapter uses a configuration file, geojni.cfg, that is put down at the time of installation, plus backward compatibility libraries. These files are in the mme.profile file, which is created and modified at installation.

The environment variable `CONFIG_FILE_PATH` must be set to the full path of the file (including the file name) and exported in the environment of an application using the adapter. For example:

```
export CONFIG_FILE_PATH=/opt/mapinfo/mapmarker/geojni.cfg
```

Backward compatibility libraries are available for both Solaris and HP-UX platforms. In addition, each platform has a JVM shared library to enable you to link your application to the JVM. The libraries for each platform and their settings are discussed below.

**Solaris Platform Libraries**

On Solaris, these are the included libraries:

- libgeoeng.11.0.so–shared library for 11.0.
- libgeoeng.11.0.a–static library for 11.0.
- libgeoeng.bc.so–shared library to use if your application was originally linked against the libgeoeng.so for a version of MapMarker prior to 8.0.
- libgeoeng.bc.80.so–shared library to use if your application was originally linked against the libgeoeng.so for MapMarker 8.0.
- libjvm.so–JVM shared library.

The installer creates a symbolic link (libgeoeng.so) that links to the v. 11.0 library (libgeoeng.11.0.so). If you need to use one of the backward compatibility libraries, you can modify this link.

The path to the JVM shared library (libjvm.so) must be part of the `LD_LIBRARY_PATH`, even if the application is built against the static geoeng library (libgeoeng.11.0.a). Solaris platforms have separate client and server objects. This is why the `-server` option is not allowed in the configuration file. Otherwise, the process of linking to the libgeoeng objects is the same as it was for version 10.0.
The `LD_LIBRARY_PATH` is created in the `mme.profile` file located in the `AdditionalTools` directory. The `mme.profile` file is created and modified by the installer at the time of installation.

**HP-UX Platform Libraries**

On the HP-UX platform, these are the included libraries:

- `libgeoeng.11.0.sl`–shared library for 11.0.
- `libgeoeng.11.0.a`–static library for 11.0.
- `libgeoeng.bc.sl`–shared library to use if your application was originally linked against the `libgeoeng.so` for a version of MapMarker prior to 8.0.
- `libgeoeng.bc.80.sl`–shared library to use if your application was originally linked against the `libgeoeng.so` for MapMarker 8.0.
- `libjvm.sl`–JVM shared library.

The installer creates a symbolic link (libgeoeng.sl) that links to the v. 11.0 library (libgeoeng.11.0.sl). If you need to use one of the backward compatibility libraries, you can modify this link.

The path to the JVM shared library (libjvm.sl) must be part of the `SHLIB_PATH`, even if the application is built against the static geoeng library (libgeoeng.11.0.a). HP-UX platforms have separate client and server shared objects. This is why the `-server` option is not allowed in the configuration file. Otherwise, the process of linking to the libgeoeng objects is the same as it was for version 10.0.

The `SHLIB_PATH` is created in the `mme.profile` file located in the `AdditionalTools` directory. The `mme.profile` file is modified at the time of installation.

On HP-UX there is another directory that must be part of the `SHLIB_PATH`. If you are using the shared library adapter, set your `SHLIB_PATH` as:

```
export SHLIB_PATH=<mapmarker lib path>:<java home>/jre/lib/PA_RISC2.0:<java home>/jre/lib/PA_RISC2.0/server:$SHLIB_PATH
```

You need both `PA_RISC2.0` and `PA_RISC2.0/server` in your `SHLIB_PATH`.

**Building HP-UX Applications Against the Shared Library (.sl) or the Static Library (.a)**

When you link an HP-UX application using either of the included libraries, the application must also be linked against the `libjvm.sl`. The following options should be used to make the application movable from one machine to another where the `libjvm.sl` or the `libgeoeng.11.0.sl` will not be in the same location: `-b +s`). This ensures that the `SHLIB_PATH` will be used to find any libraries that the application is linked against.

**Other Configuration File Settings**

The adapter configuration file contains some settings that enable you to change certain JVM settings. It also supplies country information, for those who are using multiple country geocoders, and the two coordinate systems in which point data can be returned. For more information on these settings, please see *JNI Adapter Configuration File Settings in Appendix B* on page 83.
Designing Geocoding Applications

This chapter covers the high-level design decisions involved in writing a geocoding application and offers some guidance on some common geocoding techniques that you can use to build and organize your application.

In this chapter:

- Choosing a Development Tool ........................................ 36
- Geocoding Terms ......................................................... 38
- Defining the Geocoding Model ....................................... 39
- Geocoding Application Design Overview ....................... 40
Choosing a Development Tool

When you create a geocoding application, your choice of development tool will have a direct effect on the design and deployment of your application. As you are evaluating tools, some questions to consider include:

- What are the required location-based capabilities of the application? For example, will your application need to perform only geocoding, or will you also need to provide mapping and/or routing capabilities?
- How will your application be deployed? Will users access it from the Web or will it be a desktop application?
- In what operating system(s) do you anticipate deploying your application?
- In what programming language(s) is the application to be written? Some examples include:
  - Java
  - .Net (C#, VB, C++)
  - Web services
  - XML
  - Visual Studio 6 (C/C++, VB)

MapInfo offers several geocoding products that offer different design and deployment capabilities. The product you choose must be able to meet the high-level requirements of your design.

MapInfo’s Geocoding Interoperability

The diagram below shows the interoperability between MapInfo's geocoding solutions: MapMarker, MapXtreme, and Envinsa.

The diagram indicates the level of communication between the clients, servers, and the geocoding engine. A MapXtreme client can communicate to an Envinsa server or a MapMarker server. Both servers use underlying geocoding engines.

The table below summarizes the current MapInfo geocoding solutions available.
The following sections provide a summary of some of the features and design capabilities of MapInfo’s current geocoding offerings.

**Note:** Some of these products are scheduled for retirement with the release of MapMarker Plus 11.0. You may have applications written with one of these products. For information on migrating existing applications to MapMarker Plus 11.0, please see Migrating to MapMarker Plus 11.0 in Chapter 3 on page 27.

**MapMarker Plus 11.0**

- Introduction of two separate products: Desktop and Developer. The Desktop includes the MapMarker Plus GUI and data. The Developer product has, in addition to the GUI and data, the Java API, and a JNI Adapter to run the OLE Automation API and Data Dictionary API.
- Utilizes Java geocoding engine. The advantages of using a Java geocoding engine is that it can be deployed and run on different operating systems without being recompiled.
- Provides enhanced user dictionary capabilities. The limit of five user dictionaries has been removed.
- Updates Java server geocoding features.

**MapMarker Plus 10.x**

The MapMarker Plus 10.x product is scheduled to be retired, but will still be supported with data updates through the version 11.x release schedule. Developers writing applications with MapMarker Plus 10.x:

- Create desktop or RPC client/server applications on Windows, Solaris or HP
- Desire Visual Studio 6 VB or C/C++
- Require robust geocoding features. Robust geocoding features include street or place browsing, highway exit geocoding, airport geocoding or CASS related features. CASS stands for coding accuracy support system (CASS) certification
- Perform intensive in-line process geocoding
MapMarker Plus 10.x with J Server

Developers using MapMarker Plus 10.x with MapMarker J Server:

- Build and deploy web applications on a variety of application servers (Tomcat, Weblogic, and Websphere are examples of supported application servers).
- Desire Java programming language
- Require robust geocoding features

MapMarker Plus 11.0 replaces this usage scenario. You do not need MapMarker J Server if you use MapMarker Plus 11.0. We recommend that you migrate existing applications to MapMarker Plus 11.0, and that you develop any new applications with MapMarker Plus 11.0. MapMarker Plus 10.x will be supported with data refresh releases during the v. 11.x release schedule. The MapMarker J Server API is deprecated with the release of MapMarker Plus 11.0, and both products will be retired with the release of MapMarker Plus 12.0.

MapXtreme

Developers using MapXtreme:

- Create Windows desktop or IIS web applications
- Desire .Net programming languages
- May require several location based capabilities
- Perform client/server geocoding

Envinsa

Developers using Envinsa:

- Develop geocoding applications as part of an enterprise level solution
- Create Windows, Solaris, or Linux applications
- Build and deploy web applications on a variety of application servers
- Desire .Net, Java, XML or web service programming languages
- May require several location based capabilities (e.g., reverse geocoding, which accepts an x, y position and returns address, intersection, postcode, city, or other location attribute information).

Geocoding Terms

Geocoding using MapMarker products uses a set of common terms. As you begin the design of your geocoding application, you may want to adopt the use of these terms to make labeling and defining certain elements in your application easier.

**Candidate**
Address information returned by the geocoding engine.

**Result Code**
Code that indicates how well a candidate matches the input address.
Close Match
Status that indicates whether a candidate’s ranking is considered close enough to the input address.

Address Dictionary
Binary data dictionary where address and spatial information is stored.

User Dictionary
Binary data dictionary where user address and spatial information can be stored.

Defining the Geocoding Model

The MapMarker geocoding model uses a system of relative matching. It 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, several (non-close) candidates may be returned.

Within this model, you have a certain amount of flexibility about the level of accuracy to which the address records are geocoded and how you execute the application.

Geocoding Preferences

The geocoding model contains a set of geocoding preferences that specifies the conditions under which a close match is determined. These preferences either require specific matching conditions to be true, or allow those conditions to be relaxed. The combination of preferences that are required or relaxed affects how each candidate is scored and has an impact on the number of records that are geocoded, the matching accuracy of the geocoded addresses, and the positional accuracy of the geocoded point.

For example, enabling a preference that requires a match on a specific address element may restrict or filter returned candidate information. In the MapMarker US desktop product, the default preferences include relaxing a match on street name and ZIP Code, but requiring a match on the house number. This combination of preferences provides a high geocoding success rate with few erroneous matches (false positives).

Enabling fallback preferences allows for a candidate to be geocoded to a postal centroid when a street-level match is not found. The Java API also enables you to fall back to a city centroid. Fallback preferences enable you to geocode more records, but at the sacrifice of some positional accuracy. For information on specific preferences and their uses, see Using Preferences on page 43.
Use of Geocoded Data

The intended use of the geocoded data will have a great deal of influence on how much geocoding precision capability to provide in the application. As you think about the design of your application, consider the following questions:

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

For example, perhaps you have users who need to determine the location of a new retail store and need to know the distribution of current and potential customers. In this case, geocoding as many of these customers’ addresses as possible is more important than finding an exact street match for each one. In this instance, geocoding to ZIP + 4 centroid or ZIP Code centroid is the best way to accomplish the user’s task.

On the other hand, some users may need to know where their customers are in relation to the location of something else. For example, a utility service coordinator needs to know where customers are in relation to neighborhood gas lines. In cases like this, the positional accuracy of each customer is of critical importance. Geocoding to street level with strict matching preferences would be the optimum way for these users to accomplish their tasks.

Executing the Application

Geocoding applications can be executed using two methods: automatic and interactive. Your application can perform either automatic or interactive geocoding, or use both methods, depending on the design requirements of your application.

In automatic geocoding, the matching process is pre-defined, and candidates are selected automatically. Many different preference scenarios can be used.

In interactive geocoding, the user has more control over the matching process, as well as having the capability of selecting the match candidate from a list. Match restrictions can be relaxed, and falling back to a postal-centroid level of accuracy is also an option.

Geocoding Application Design Overview

All geocoding applications use a common approach that is made up of a number of tasks:

- Select a geocode type
- Build input address
- Set geocoding preferences
- Send a request
- Verify response
- Use candidate information
When you design your geocoding application, focus on each of the steps to determine the overall flow of the geocoding application.

Types of Geocoding

Two of the most common types of geocoding are street and postal. The type of geocoding you want your application to perform depends on the needs of your users and how much positional accuracy they require.

Street Geocoding

In street geocoding, candidate coordinates are interpolated along a street segment. Geocoding to street level achieves the highest point accuracy, unless you have a point-level user dictionary. This image shows how a candidate’s point, 18 Long Branch Ave, is interpolated along a street segment. The street’s range is from 24 to 14. The geocoder interpolates the point’s position based on the candidate’s house number.

Postal Geocoding

Postal level geocoding is faster but not as geographically accurate. Urban and rural accuracy can produce different results. This image shows the location of postal centroids as red dots. You can see why they would be less accurate than points geocoded to street level. (The centroid positions are delivery point weighted.) This image does not show U.S. ZIP + 4 centroid values; ZIP + 4 centroids have better postal accuracy than ZIP Code centroids. The blue polygons in the image are postal boundaries. Rural area polygons cover larger area and therefore are less accurate than urban areas.
Building an Input Address

The input address is the address to be geocoded. MapInfo uses two different kinds of input address objects: one is country specific; the other can be used for multiple countries (generic address).

Some geocoding APIs use different naming conventions. This table shows the correspondences between the address elements in the input address object for generic addresses and U.S.-specific addresses.

<table>
<thead>
<tr>
<th>Generic Address</th>
<th>Country-Specific Address (USA)</th>
<th>Example (USA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building or Place Name</td>
<td>Firm</td>
<td>MapInfo Corporation</td>
</tr>
<tr>
<td>Street or Main Address</td>
<td>Street</td>
<td>1 Global View</td>
</tr>
<tr>
<td>Municipality or AreaName3</td>
<td>City</td>
<td>Troy</td>
</tr>
<tr>
<td>Country Subdivision or AreaName1</td>
<td>State</td>
<td>NY</td>
</tr>
<tr>
<td>Primary Postcode or Postcode1</td>
<td>ZIP Code</td>
<td>12180</td>
</tr>
<tr>
<td>Secondary Postcode or Postcode2</td>
<td>ZIP + 4</td>
<td>8337</td>
</tr>
<tr>
<td>Municipality Subdivision or AreaName4</td>
<td>Urbanization</td>
<td>Parc Sabana (used in Puerto Rico)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Puerto Rico is U.S. only. Leave blank if not geocoding Puerto Rican addresses.</td>
</tr>
<tr>
<td>*Country (required for generic)</td>
<td>Set by default</td>
<td>USA</td>
</tr>
</tbody>
</table>

*The Country property is required when using a generic input address object.

For addresses in the United States, there are a number of input requirements. These are specific address elements that must be present so that the application can geocode the address.
Using Preferences

As explained earlier in this chapter, preferences enable you to define the conditions under which a close match is determined. The geocoding application will match addresses according to these conditions. The conditions can be very precise to ensure more exact address matches and a high degree of positional accuracy, or they can be relaxed to ensure that as many records as possible are geocoded.

MapInfo uses two types of preference objects: country specific, and a generic or universal, which can be used for multiple countries. Some APIs use different naming conventions. This table shows the correspondences between the generic preference names and the USA-specific preference names.

<table>
<thead>
<tr>
<th>Desired Geocoding Level</th>
<th>Required Input Fields</th>
<th>Other Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Street</td>
<td>Street and ZIP Code, or street, city, and state</td>
<td>Country required in generic address structure.</td>
</tr>
<tr>
<td>Street Intersection</td>
<td>Street and ZIP Code, or street, city, and state Street 1 and street 2 names must be separated by double ampersands “&amp;&amp;” (e.g., “Global View &amp;&amp; Jordan Road”, “42nd &amp;&amp; Park”)</td>
<td></td>
</tr>
<tr>
<td>Postal Centroid</td>
<td>ZIP Code and/or ZIP + 4</td>
<td>Country required in generic address structure</td>
</tr>
<tr>
<td>Geo</td>
<td>City and State</td>
<td>Available through Java API</td>
</tr>
</tbody>
</table>

### Using Preferences

<table>
<thead>
<tr>
<th>Generic</th>
<th>Country – USA</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MustMatchAddressNumber</td>
<td>Same as generic</td>
<td>When enabled, candidates must match the input address number to be considered close matches.</td>
</tr>
<tr>
<td>MustMatchMainAddress</td>
<td>MustMatchStreet</td>
<td>When enabled, candidates must match the input main address to be considered close matches.</td>
</tr>
<tr>
<td>MustMatchMunicipality or MustMatchAreaName3</td>
<td>MustMatchCity</td>
<td>When enabled, candidates must match the input city to be considered close matches.</td>
</tr>
<tr>
<td>MustMatchPostalCode</td>
<td>MustMatchZipcode</td>
<td>When enabled, candidates must match the input postcode to be considered close matches.</td>
</tr>
<tr>
<td>CloseMatchesOnly</td>
<td>Same as generic</td>
<td>When enabled, only close match candidates are returned.</td>
</tr>
<tr>
<td>FallbackToPostal</td>
<td>Same as generic</td>
<td>When enabled, will return a postal centroid when a close match on the street is not found.</td>
</tr>
</tbody>
</table>
Preference Usage Examples

Different combinations of preferences will return different candidate results. These examples illustrate how the candidates list for an input address changes when you enable additional preferences.

Matching on Address Number

When you enable the Must Match Address Number preference, the input address:

350 Ocean Drive
Key Biscayne, FL 33149

returns five possible candidates. Among these candidates, one candidate is a close match.

Matching on Address Number; Close Matches Only

When you also enable the Close Matches Only preference in addition to the Must Match Address Number preference, the input address:

350 Ocean Drive
Key Biscayne, FL 33149

returns only the close match candidate. When the Close Matches Only preference is enabled, only the close match candidate is returned.

Matching on Address Number, Close Match Only, and Falling Back to Postal

When you also enable the Fallback to Postal preference both the Must Match Address Number and Close Matches Only preferences, and also enable the Fallback to Postal preference, the input address:

3050 Ocean Drive
Key Biscayne, FL 33149

returns a postal candidate instead of a street candidate.

The reason only a postal candidate is returned is that the Must Match Address Number preference is enabled, and the input address number does not match the close match candidate’s address number. When this happens, and the Fallback to Postal preference is enabled, a postal candidate is returned.
**Sending a Request**

The geocode request contains all the information needed for the application to geocode an address. The geocode type, the input address, the geocoding preferences, and, if using a server, the URL or IP address comprise the information in the geocode request.

**Verifying a Response**

The response you receive back from a geocode request contains the following information:

- Exceptions
- Returned candidate count
- Candidate result code

Exception error handling checks for system-level failures. Some examples of these include:

- Invalid server URL or IP address
- Server is down
- Invalid address dictionary path

When an exception occurs, the geocoding process should be halted until the error is corrected.

The response to a successful geocode request also contains a returned candidate count. In a returned candidate count, please keep in mind the following:

- Possible candidates verses returned candidates
- Use returned value when looping through candidates
- Possible to receive no candidates

Finally, every candidate will have a result code. The result code indicates how precisely the candidate matches the input address.

**Using Geocode Result Information**

The result code is made up of 10 alphanumeric characters. For example, a result code might look like this: S5HPNTSCZA.

Each character in the code describes something about the geocoded point: the type of geocoding, the positional accuracy of the point, the quality of the match of specific address elements, and the dictionary that the candidate matched on.

This section provides a brief summary of what each character in the code indicates. For detailed information on MapMarker result codes, please see the Result Codes chapter in the MapMarker Plus Desktop User Guide.

**Geocode Type**

The first character in the result code indicates the geocoding type. The first character can be one of the following characters:

- S – Street Type
- Z – ZIP or postal code type
- G – geographic type
**Positional Accuracy**

The second character in the code reflects the positional accuracy of the candidate’s point. The character can be the numbers 0-6, or the letter X.

- 6 – Indicates highest accuracy (point zip, building)
- 5 – Street interpolated point
- 4 – Street centroid point
- 3,2,1 – Postal centroid point
- 0 – No geometry available (extremely rare)
- X – Intersection point

**Address Elements**

The characters in the third through the ninth positions in the result code describe the matches in the different elements in the address.

- The third position describes **H**ouse or address number match (e.g. 115)
- The fourth position describes street **P**refix directional match (e.g. North)
- The fifth position describes street **N**ame or main address match (e.g. School)
- The sixth position describes street **T**ype match (e.g. Avenue)
- The seventh position describes street **S**uffix Directional match (e.g. NE)
- The eighth position describes **C**ity match (e.g. Miami)
- The ninth position describes **Z**ip or postal match (e.g. 80302)

**Dictionary Type**

The tenth and last character in the code describes the candidate’s dictionary type: A or U.

- A – Address dictionary
- U – User dictionary

**Input Address Conflicts**

A dash "-" in the result code indicates a conflict with the input address. For example, in the result code: S5HPNTSC-A the Z (ZIP Code) is in conflict with the input address.
Using the MapMarker Plus Java API

This chapter builds on the design principles discussed in the previous chapter and shows you how to build a geocoding application using the MapMarker Plus Java API. It also discusses how to deploy your application, as well as some of the geocoding features available in the API such as geographic centroid geocoding, and street or place name browsing.

In this section:

- Overview ...................................................... 48
- Developing Geocoding Applications in Java ............... 48
- Browsing Addresses ......................................... 58
- Geocoding to City Centroids ................................. 59
- Geocoding to Highway Exits .............................. 61
- Geocoding to Airports .................................... 62
Overview

Geocoding turns ordinary data records containing address information into geographic objects that can be displayed on a map. This helps you to visualize the relationships between your data. You can display your geocoded records in MapInfo MapXtreme and add other map objects to give your data a geographic reference. You can then combine MapMarker, MapXtreme, and the MapInfo Routing J Server to create custom solutions to solve the needs of your organization.

Using the MapMarker Plus Java USA API

The MapMarker Plus Java USA API is used to create geocoding applications specifically for U.S. addresses. The API uses common U.S. address terminology and is a good choice for developers who are geocoding U.S. addresses.

Using the MapMarker Generic Java API

The MapMarker Generic Java API is used to create international geocoding applications. The Generic API can be used with any MapMarker country geocoder. The API uses common universal address terminology and is a good choice for a developer whose requirement is to geocode addresses from many different countries. For more information on deploying MapMarker Plus as a servlet along with other MapMarker countries see Integration with MapMarker for Other Countries on page 69.

For detailed information about all classes, refer to the API documentation for MapMarker Plus USA. It is located in the SDK\MapMarker-4.0.0\docs folder under the directory where you installed the product.

Developing Geocoding Applications in Java

Use either the MapMarker Plus USA Java API or the MapMarker Generic API to develop custom geocoding applications in Java. See Using the MapMarker Plus Java USA API on page 48 and Using the MapMarker Generic Java API on page 48 for more information on which API to use for your particular situation.

Creating a Geocoding Application in Java

Make sure that your application contains these import statements:

```java
/* MapMarker Plus USA Java API */
import com.mapinfo.mapmarker.user.MMJEngine;
import com.mapinfo.mapmarker.user.ClientGeocodeResponse;
import com.mapinfo.mapmarker.USA.USA_GeocodeConstraints;
import com.mapinfo.mapmarker.USA.USA_UserCandidateAddress;
import com.mapinfo.mapmarker.USA.USA_UserInputAddress;

/* MapMarker Generic Java API */
```
import com.mapinfo.mapmarker.user.MMJEngine;
import com.mapinfo.mapmarker.client.ClientGeocodeResponse;
import com.mapinfo.mapmarker.IGeocodeConstraints;
import com.mapinfo.mapmarker.common.AddressImpl;
import com.mapinfo.mapmarker.common.Address;

Make sure that the following jar files are in your classpath. You can find these files in the MapMarker Plus folder under the directory where you installed the product:

- mmj.jar
- mmj_usa.jar
- miscp.jar
- mmjclient.jar
- mmjclient_usa.jar
- micsys.jar
- miutil.jar

Also, a copy of the USA_DataManagerSettings.properties file must be in your classpath with the correct location of the MapMarker data.

### Setting the Input Address

The first step in creating a geocoding application is setting the input address. Here is an example of how to set the input address if you were using the MapMarker Plus USA Java API.

```java
/* Input address example */
String addr = "1 GLOBAL VIEW";
String postcode = "12180";
String city = "TROY";
String state = "NY";

USA_UserInputAddress inpAddr = new USA_UserInputAddress();

/* Set the input Address */
inpAddr.setStreet(addr); /* 'street' */
inpAddr.setCity(city); /* 'city' */
inpAddr.setState(state); /* 'state' */
inpAddr.setZipcode(postcode); /* 'zipcode' */
```

If you were using the MapMarker Generic API, you would use the Address class implemented with AddressImpl.

```java
Address inpAddr = new AddressImpl();

/* Set the input Address */
inpAddr.setCountry("USA");
inpAddr.setMainAddress(addr); /* street */
inpAddr.setAreaName3(areaname3); /* city */
inpAddr.setAreaName1(areaname1); /* state */
inpAddr.setPostCode1(postcode); /* zipcode */
```

For detailed information on all classes, refer to the API documentation for MapMarker Plus USA. It is located in the SDK\MapMarker-4.0.0\docs folder under the directory where you installed the product.
Setting Geocoding Constraints

Next you need to set the geocoding constraints (or preferences). If no constraints are explicitly provided, MapMarker uses default settings. Here is an example of how to set constraints using the MapMarker Plus USA Java API.

```java
USA_GeocodeConstraints geoCon = new USA_GeocodeConstraints();
    /* Geocode Constraints:
    * Review Java docs for a complete constraint list and
    * default values
    */
    geoCon.setMustMatchAddressNumber(true);
    geoCon.setMaxCandidates(1);
    geoCon.setReturnCloseCandidatesOnly(true);
    geoCon.setFallbackToPostal(false);
    geoCon.setCassMode(true);
```

If you are using the MapMarker Generic API you would use the GeocodeConstraints class:

```java
GeocodeConstraints=new GeocodeConstraints();
    constraints.setMustMatchAddressNumber(true);
    constraints.setMaxCandidates(1);
    constraints.setReturnCloseCandidatesOnly(true);
    constraints.setFallbackToPostal(true);
    constraints.setCustomString(USA_GeocodeConstraints.KEY_CASSRULES, "true");
```

Geocoding constraints affect the conditions under which MapMarker Plus attempts to match a record. Changing settings can affect the time in which MapMarker takes to geocode a record. Choose matching conditions to fit your needs. Generic (GeocodeConstraints) and U.S. (USA_GeocodeConstraints) constraints are described as follows.

### GeocodeConstraints

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_CLIENT_CRS</td>
<td>String describing the client coordinate system. Accepted values include EPSG SRS Names as well as MAPINFO MAPBASIC Projection String SRS Names. (see com.mapinfo.coordsys.CoordSys). Default uses WGS 84.</td>
<td>&quot;epsg:4326&quot;</td>
</tr>
<tr>
<td>KEY_CLOSEMATCHESONLY</td>
<td>Only close matches are returned.</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>KEY_MAXCANDIDATES</td>
<td>Max candidates to be returned (use &quot;-1&quot; to return all candidates)</td>
<td>&quot;3&quot;</td>
</tr>
<tr>
<td>KEY_MAXRANGES</td>
<td>Max ranges per candidate to be returned (use &quot;-1&quot; to return all ranges)</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>KEY_MAXRANGEUNITS</td>
<td>Max units per range to be returned</td>
<td>&quot;0&quot;</td>
</tr>
<tr>
<td>KEY_MUST_MATCH_ADDRNUM</td>
<td>Only candidates matching address number are considered close.</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>KEY_MUST_MATCH_MAINADDR</td>
<td>Only candidates matching the main address are considered close.</td>
<td>&quot;false&quot;</td>
</tr>
</tbody>
</table>
USA_GeocodeConstraints

USA_GeocodeConstraints extends the GeocodeConstraints object. It provides both a default and copy constructor (no arguments or a GeocodeConstraints object as the argument). It defines the following local string constants:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_MUST_MATCH_AREA1</td>
<td>Only candidates matching a country subdivision are considered close (e.g., State for USA or Province for Canada).</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_MUST_MATCH_AREA3</td>
<td>Only candidates matching a city or town are considered close.</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_MUST_MATCH_POSTAL</td>
<td>Only candidates matching postal code are considered close.</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_MUST_MATCH_INPUT</td>
<td>Only candidates matching all input areas (where MUST_MATCH is defined) are considered close.</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_FALLBACK_TO_POSTAL</td>
<td>If no close street level candidates found return postal centroid for the input address.</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_CORNEROFFSET</td>
<td>Defines the position of the geocoded point with respect to the corner.</td>
<td>“7”</td>
</tr>
<tr>
<td>KEY_CORNEROFFSETUNITS</td>
<td>The units used in KEY_CORNEROFFSET.</td>
<td>“m” (meters)</td>
</tr>
<tr>
<td>KEY_STREETOFFSET</td>
<td>Defines the position of the geocoded point with respect to the centerline of the street.</td>
<td>“7”</td>
</tr>
<tr>
<td>KEY_STREETOFFSETUNITS</td>
<td>The units used in KEY_STREETOFFSET.</td>
<td>“m” (meters)</td>
</tr>
</tbody>
</table>

USA_GeocodeConstraints

USA_GeocodeConstraints extends the GeocodeConstraints object. It provides both a default and copy constructor (no arguments or a GeocodeConstraints object as the argument). It defines the following local string constants:
Setting the Coordinate System

A coordinate system is a recognized reference system for the unique location of a point in space. Cartesian (planar) and Geodetic (geographical) coordinates are examples of reference systems based on Euclidean geometry.

MapMarker Plus uses all of the coordinate systems supported by MapXtreme, including systems recognized by the European Petroleum Survey Group (EPSG).

Coordinate systems are set through the USA_GeocodeConstraints method setClientCoordinateSystem. This class extends the GeocodeConstraints class, which implements IGeocodeConstraints.

```
USA_GeocodeConstraints constraints = new USA_GeocodeConstraints();
constraints.setClientCoordinateSystem(CoordSys.longLatWGS84);
```

If a coordinate system is not defined, then MapMarker Plus uses the Longitude / Latitude (WGS 84) coordinate system by default.

The following table lists common coordinate systems used in the United States:

<table>
<thead>
<tr>
<th>Key</th>
<th>Base Equivalent</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_EXPANDED_SEARCH_LIMIT_TO_STATE</td>
<td></td>
<td>Do not cross state boundaries when doing an expanded search.</td>
<td>“true”</td>
</tr>
<tr>
<td>KEY_CASSRULES</td>
<td></td>
<td>Turn on CASS mode.</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_CASS_DPV</td>
<td></td>
<td>Turn on DPV mode.</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_ALTERNATE_LOOKUP</td>
<td></td>
<td>Determines whether the preferred lookup is to look for streets first or places first.</td>
<td>STREET_FIRST</td>
</tr>
<tr>
<td>KEY_LACSLINK</td>
<td></td>
<td>Turn on LACSLink mode.</td>
<td>“false”</td>
</tr>
</tbody>
</table>

Geocoding

Now that you have set the input address and have set the geocoding constraints, you are ready to geocode. Use the geocode method of the MMJEngine class as shown in this example. This class is used for geocoding with either the MapMarker Plus USA Java API or the MapMarker Generic Java API. To geocode using the MMJClient class see Creating a Web Geocoding Client in Java on page 54.
/* Geocode Address */
MMJEngine engine = new MMJEngine();
ClientGeocodeResponse geoRes =
  engine.geocode(com.mapinfo.mapmarker.user.MapMarkerJavaAPI.Geocode_Type_Address, inpAddr, geoCon);

Getting Candidates

First, you can check to see how many candidates were found. You can do this by looking at the totalPossibleCandidates property of the ClientGeocodeResponse object.

```java
if (geoRes.totalPossibleCandidates() >0) {
    /* [. . . ] */
}
```

Out of all the candidates found, you can also see how many were close matches. You can do this by looking at the totalPossibleCloseMatchCandidates property and the of the ClientGeocodeResponse object.

```java
if (geoRes.totalPossibleCloseMatchCandidates() >0) {
    /* [. . . ] */
}
```

MapMarker Plus only returns a subset of the total possible candidates that it has found. As a default MapMarker returns the first three candidates that are found. If you would like to see more of the candidates, call setMaxCandidates(x) where x is the maximum number of candidates desired.

You can check to see how many candidates were actually returned by looking at the candidateCount property of the ClientGeocodeResponse object.

```java
if (geoRes.candidateCount() >0) {
    /* [. . . ] */
}
```

Now you can use the USA_UserCandidateAddress to obtain detailed candidate information.

```java
/* Get candidate count */
int candCount = geoRes.candidateCount();
/*
 * Get returned candidates
 */
if (candCount > 0)
{
    for (int i=0; i < candCount; i++)
    {
        /*Get candidate information */
        USA_UserCandidateAddress usaCand = new USA_UserCandidateAddress
(geoRes.candidateAt(i));
        System.out.println(usaCand.toString());
        System.out.println(usaCand.getLocation().toString());
    }
}
Getting the Result Code

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 tells how precisely MapMarker matched each address component. The result code is obtained using the getPrecisionCode method of the USA_UserCandidateAddress object. Here is an example of how to get the precision code:

```java
String resultCode = usacand.getGeoResult();
```

For more information on interpreting the geocoding results, see the Result Codes chapter in the MapMarker Plus Desktop User Guide.

Creating a Web Geocoding Client in Java

If you have deployed MapMarker Plus as a servlet, you need to develop a web client application to send geocoding requests to the server. Follow all of the steps under Creating a Geocoding Application in Java on page 48, except use the MMJClient class instead of the MMJEngine class. The MMJClient geocode method requires the URL of the servlet.

```java
/* Geocode Address */
String url = new String("http://mapinfo:8095/mapmarker40/servlet/mapmarker")
MMJClient m_client;
ClientGeocodeResponse geoRes = m_client.geocode(0, inpAddr, geoCon, url);
```

Make sure that your application contains these import statements

```java
/* MapMarker Plus USA Java API */
import com.mapinfo.mapmarker.user.MMJClient;
import com.mapinfo.mapmarker.user.ClientGeocodeResponse;
import com.mapinfo.mapmarker.user.MMJClient;
import com.mapinfo.mapmarker.user.ClientGeocodeResponse;
import com.mapinfo.mapmarker.USA.USA_GeocodeConstraints;
import com.mapinfo.mapmarker.IGeocodeConstraints;
import com.mapinfo.mapmarker.USA.USA_UserCandidateAddress;
import com.mapinfo.mapmarker.USA.USA_UserInputAddress;

/* MapMarker Generic Java API */
import com.mapinfo.mapmarker.client.MMJClient;
import com.mapinfo.mapmarker.client.ClientGeocodeResponse;
import com.mapinfo.mapmarker.client.MMJClient;
import com.mapinfo.mapmarker.client.ClientGeocodeResponse;
import com.mapinfo.mapmarker.GeocodeConstraints;
import com.mapinfo.mapmarker.IGeocodeConstraints;
import com.mapinfo.mapmarker.common.AddressImpl;
import com.mapinfo.mapmarker.common.Address;
```

Make sure that these jar files are in your classpath. You can find these files in the SDK\MapMarker-4.0.0\lib\client and SDK\MapMarker-4.0.0\lib\common folders under the directory where you installed the product.

- mmjclient.jar
- mmjclient_usa.jar
- miutil.jar
- micsys.jar
- jdom.jar
- commons-logging.jar
- xercesImpl.jar
- xmlParserAPIs.jar
Also make sure that the path to encoding-map.xml is in your classpath. You can find this file in the SDK\MapMarker-4.0.0\lib\client folder under the directory where you installed the product.

Sample Code

The following usage example is a web client created using the MapMarker Plus USA API.

```java
/* MapInfo Imports */
import com.mapinfo.mapmarker.USA.USA_GeocodeConstraints;
import com.mapinfo.mapmarker.user.ClientGeocodeResponse;
import com.mapinfo.mapmarker.user.MMJClient;
import com.mapinfo.mapmarker.USA.USA_GeocodeConstraints;
import com.mapinfo.mapmarker.USA.USA_UserCandidateAddress;
import com.mapinfo.mapmarker.USA.USA_UserInputAddress;

public class MMJ_US_Example {

    private void geocode() {
        /* Set server url*/
        String url = "http://localhost:8095/mapmarker40/servlet/mapmarker";

        /* Input address example */
        String addr = "1 GLOBAL VIEW";
        String city = "TROY";
        String state = "NY";
        String postal = "12180";

        USA_GeocodeConstraints geoCon = new USA_GeocodeConstraints();

        /* Geocode Constraints:
        * Review Java docs for a complete constraint list and
        * default values
        */
        geoCon.setMustMatchAddressNumber(true);
        geoCon.setMaxCandidates(1);
        geoCon.setReturnCloseCandidatesOnly(true);
        geoCon.setFallbackToPostal(false);
        geoCon.setCassMode(true);

        USA_UserCandidateAddress canAddr = null;
        USA_UserInputAddress inpAddr = new USA_UserInputAddress();

        /* Set the input Address */
        inpAddr.setStreet(addr); /*street*/
        inpAddr.setCity(city); /*city*/
        inpAddr.setState(state); /*state*/
        inpAddr.setZipcode(postal); /*zipcode*/

        /*Geocode Address */
        ClientGeocodeResponse geoRes = MMJClient.geocode(
            MMJClient.GEOCODE_TYPE_ADDRESS, inpAddr, geoCon, url);

        /*Get candidate count*/
        int candCount = geoRes.candidateCount();

        /*
        */
        /*Get returned candidates
        */
    }
}
```
if(candCount > 0)
{
  for (int i=0; i< candCount; i++)
  {
    /*Get candidate information */
    CandidateAddress candAddr = geoRes.candidateAt(i);
    USA_UserCandidateAddress usaCand = new
        USA_UserCandidateAddress(candAddr);
    System.out.println(usaCand);
    System.out.println(usaCand.getLocation());
  }
} /* End */

public static void main(String[] args)
{
    MMJ_US_Example geocodeTest = new MMJ_US_Example();
    geocodeTest.geocode();
}

The following usage example is a web client created using the MapMarker Generic API.

/* MapInfo Imports */
import com.mapinfo.mapmarker.CandidateAddress;
import com.mapinfo.mapmarker.user.ClientGeocodeResponse;
import com.mapinfo.mapmarker.user.MMJClient;
import com.mapinfo.mapmarker.GeocodeConstraints;
import com.mapinfo.mapmarker.IGeocodeConstraints;
import com.mapinfo.mapmarker.common.AddressImpl;
import com.mapinfo.mapmarker.common.Address;

public class MMJ_Generic_Example
{
  private void geocodeGenericAddress() {
    /* Set server url*/
    String url =
        "http://localhost:8095/mapmarker40/servlet/mapmarker";

    /* Input Address Example*/
    String addr = "1 GLOBAL VIEW";
    String city = "TROY";
    String state = "NY";
    String postal = "12180";

    GeocodeConstraints geoCon = new GeocodeConstraints();

    /* Geocode Constraints:
    * Review Java docs for a complete constraint list
    * and default values
    */
    geoCon.setMustMatchAddressNumber(true);
    geoCon.setMaxCandidates(1);
    geoCon.setReturnCloseCandidatesOnly(true);
geoCon.setFallbackToPostal(false);
geoCon.setCustomString(USA_GeocodeConstraints.KEY_CASSRULES, "true");

Address inpAddr = new AddressImpl();

/* Set the input Address */
inpAddr.setCountry("USA");
inpAddr.setMainAddress(addr); /*street*/
inpAddr.setAreaName3(areaName3); /*city*/
inpAddr.setAreaName1(areaName1); /*state*/
inpAddr.setPostCode1(postcode); /*zipcode*/

/* Geocode Address */
ClientGeocodeResponse geoRes =
MMJClient.geocode(MapMarkerJavaAPI.GEOCODE_TYPE_ADDRESS, inpAddr,
geoCon, url);

/* Get candidate count */
int candCount = geoRes.candidateCount();

/*
* Get returned candidates
*/
if(candCount > 0)
{
    for (int i=0; i < candCount; i++) {
        /** Get candidate information
         * Review Java docs for a complete list of
         * candidate information
         */
        CandidateAddress canAddr = geoRes.candidate(i);
        System.out.println(canAddr);
        System.out.println(canAddr.getLocation());
        System.out.println("Precision Code:" + canAddr.getPrecisionCode());
        System.out.println();

        /** Get candidate range information
         * Review Java docs for a complete list of candidate range
         * information
         */
        for (int k=0;
            k<canAddr.getNumberOfCandidateRangesFound();k++)
        {
            CandidateRange range = can.getRangeAt (k);
            System.out.println("Range Number:" + (k+1));
            System.out.println("Street Side:" +
            range.getLeftRightIndicator ());
            System.out.println("Low Number:" +
            range.getLowAddress());
            System.out.println("High Number:" +
            range.getHighAddress());
            System.out.println();
        }
    }
}
Browsing Addresses

Browsing enables you to retrieve a list of possible address candidates from an address dictionary using a partial street or firm name with a city/state and/or ZIP Code as input. No geocoding occurs when you browse, but you are able to view the list of possible candidates.

To browse an address, use MapMarker JavaAPI.GEOCODE_TYPE_BROWSE in the call. The candidates returned will be in the form of a UserCandidateAddress object, which will hold the street name information.

/* Input address example */

USA_UserInputAddress inpAddr = new USA_UserInputAddress();
inpAddr.setStreet("G");
inpAddr.setZipcode("12180");

If you were using the MapMarker Generic API, you would use the AddressImpl class.

AddressImpl inpAddr = new AddressImpl();
inAddr.setMainAddress("G");
inAddr.setPostCode1("12180");
inAddr.setCountry("USA");

For detailed information on all classes, refer to the API documentation for MapMarker Plus USA. It is located in the SDK\MapMarker-4.0.0\docs folder under the directory where you installed the product.

Use the geocode method of the MMJEngine class as shown in this example. This class is used forPost with either the MapMarker Plus USA Java API or the MapMarker Generic Java API. To geocode using the MMJClient class see Creating a Web Geocoding Client in Java on page 54.

    /* Geocode Address */
    MMJEngine engine = new MMJEngine();
    ClientGeocodeResponse geoRes = null;
    geoRes = engine.geocode(MapMarkerJavaAPI.GEOCODE_TYPE_BROWSE, inpAddr, geoCon);
All of the streets that begin with "G" in the ZIP Code "12180" will be returned. Now you can use the USA_UserCandidateRange to obtain detailed candidate information.

```java
/* Get candidate count */
int candCount = geoRes.candidateCount();
/*
 * Get returned candidates
 */
if (candCount > 0)
{
    for (int i=0; i < candCount; i++)
    {
        /*Get candidate information */
        USA_UserCandidateAddress candAddr = new USA_UserCandidateAddress(geoRes.candidateAt(i));
        if (candAddr.getNumberOfCandidateRanges() > 0)
        {
            USA_UserCandidateRange candRange = new USA_UserCandidateRange(candAddr.getRangeAt(0));
            System.out.println("Candidate Range :" + candRange);
        }
    }
}
```

### Geocoding to City Centroids

A new feature for MapMarker Plus 11.0 is the ability to geocode records to a geographic centroid. For U.S. addresses, cities are the geographic centroid. This feature is available through the MapMarker Java USA API. Please note that it is not available in the MapMarker Plus Desktop product.

The source data for the city centroids comes from MapInfo's data product *2003 CityInfoAll*, which contains over 160,000 records for cities and named places in the United States and Puerto Rico compiled from various government agencies.

To geocode your records to a geographic (city) centroid, use MapMarkerJavaAPI.GEOCODE_TYPE_GEOGRAPHIC_CENTROID.

```java
/* Input address example */
USA_UserInputAddress inpAddr = new USA_UserInputAddress();
inpAddr.setCity("Albany");
inpAddr.setState("NY");

If you were using the MapMarker Generic API, you would use the AddressImpl class.

AddressImpl inpAddr = new AddressImpl();
inAddr.setAreaName3("Albany");
inAddr.setAreaName1("NY");
inAddr.setCountry("USA");
```
For detailed information on all classes, refer to the API documentation for MapMarker Plus USA. It is located in the SDK\MapMarker-4.0.0\docs folder under the directory where you installed the product.

To geocode, use the geocode method of the MMJEngine class as shown in this example. This class is used for geocoding with either the MapMarker Plus USA Java API or the MapMarker Generic Java API. To geocode using the MMJClient class see Creating a Web Geocoding Client in Java on page 54.

```java
/* Geocode Address */
MMJEngine engine = new MMJEngine();
ClientGeocodeResponse geoRes = null;
geoRes =
    engine.geocode(MapMarkerJavaAPI.GEOCODE_TYPE_GEOGRAPHIC_CENTROID, inpAddr, geoCon);

You can also specify a preference to “fall back to geographic centroid” from a street-level geocode.

```java
/*Set Fallback to Geographic */
USA_GeocodeConstraints geoCon = new USA_GeocodeConstraints();
geoCon.setFallbackToGeographic(geoRes = engine.geocode(MapMarkerJavaAPI.GEOCODE_TYPE_ADDRESS, inpAddr, geoCon));

After the geocoding call you can use the USA_UserCandidateAddress to obtain detailed candidate information.

```java
/* Get candidate count */
int candCount = geoRes.candidateCount();

/* Get returned candidates*/
if (candCount > 0)
{
    for (int i=0; i < candCount; i++)
    {
        /*Get candidate information */
        USA_UserCandidateAddress candAddr = new
        USA_UserCandidateAddress(geoRes.candidateAt(i));
    }
}

City centroid candidates may contain values for areaNames 1-3 (state, county, city), census block, rank, and location. The georresult code is always G3 in the USA with the current data source.

For deciding close matches, preference is given to the candidate where the spelling of the city is closer to the input address. If more than one geographic area with the same name exists (e.g., Springfield, VA exists in eight different counties), preference is given to the higher ranking city.

The rank of the city is determined from the source data. Each record is ranked from 0 to 7, with 0 being the most populous/important cities. A rank of 7 indicates a neighborhood level geographic area (“South Troy NY”). If two close candidates are compared, the one with the lower-numbered rank (bigger city) is kept as a close match and the other candidate is demoted to a non-close match.
Geocoding to Highway Exits

Geocoding to highway exits using the MapMarker Java API returns candidates that are geocoded to ZIP Code centroid accuracy. You must supply the highway exit information, and the state abbreviation. This input information is put into an InputAddress object. It is parsed into two fields:

- mainAddress field—holds the highway exit information.
- state field—holds the state abbreviation.

To geocode a highway exit, use MapMarkerJavaAPI.GEOCODE_TYPE_HIGHWAY_EXITS

/* Input address example */

USA_UserInputAddress inpAddr = new USA_UserInputAddress();
inpAddr.setStreet("I-90 Exit 8");
inpAddr.setState("NY");

If you were using the MapMarker Generic API, you would use the AddressImpl class.

AddressImpl inpAddr = new AddressImpl();
inpAddr.setMainAddress("I-90 Exit 8");
inpAddr.setAreaName1("NY");
inpAddr.setCountry("USA");

For detailed information on all classes, refer to the API documentation for MapMarker Plus USA.

To geocode, use the geocode method of the MMJEngine class as shown in this example. This class is used for geocoding with either the MapMarker Plus USA Java API or the MapMarker Generic Java API. To geocode using the MMJClient class see Creating a Web Geocoding Client in Java on page 54.

/* Geocode Address */

MMJEngine engine = new MMJEngine();
ClientGeocodeResponse geoRes = null;
geoRes = engine.geocode(MapMarkerJavaAPI.GEOCODE_TYPE_HIGHWAY_EXITS,
inpAddr, geoCon);

After the geocoding call you can use the USA_HighwayExitCandidateAddress to obtain detailed candidate information.

/* Get candidate count */
int candCount = geoRes.candidateCount();

/* Get returned candidates*/
if (candCount > 0)
{
    for (int i=0; i < candCount; i++)
    {
        /*Get candidate information */
        USA_HighwayExitCandidateAddress candAddr = new
        USA_HighwayExitCandidateAddress (geoRes.candidateAt(i));
    }
}
Geocoding to Airports

Make sure that your application contains these import statements.

```java
import com.mapinfo.mapmarker.USA.USA_UserAirportCandidateAddress;
import com.mapinfo.mapmarker.USA.USA_CustomGeocoder;
```

Finding Airports by State

If you want to find all of the airports in a given state use the geocoding type:

```java
USA_CustomGeocoder.GEOCODE_TYPE_AIRPORTS_BY_STATE.
```

```java
/* Input address example */
String state = "NY";
USA_UserInputAddress inpAddr = new USA_UserInputAddress();
/* Set the input Address */
inpAddr.setState(state); /* State 'NY' */
```

If you were using the MapMarker Core API, you would use the AddressImpl class.

```java
AddressImpl inpAddr = new AddressImpl();
/* Set the input Address */
String country="USA"
inpAddr.setAreaName1(state); /* state 'NY' */
inpAddr.setCountry(country); /* country 'USA' */
```

For detailed information on all classes, refer to the API documentation for MapMarker Plus USA. It is located in the SDK\MapMarker-4.0.0\docs folder under the directory where you installed the product.

Use the geocode method of the MMJEngine class as shown in this example. This class is used for geocoding with either the MapMarker Plus USA Java API or the MapMarker Core Java API. To geocode using the MMJClient class see Creating a Web Geocoding Client in Java on page 54.

```java
/* Geocode Address */
MMJEngine engine = new MMJEngine();
ClientGeocodeResponse geoRes = null;
geoRes = engine.geocode(USA_CustomGeocoder.GEOCODE_TYPE_AIRPORTS_BY_STATE,
inpAddr, geoCon);
```

All of the known airports located in the specified state will be returned. Now you can use the USA_UserAirportCandidateAddress to obtain detailed candidate information.

```java
/* Get candidate count */
int candCount = geoRes.candidateCount();
/*
* Get returned candidates
*/
if (candCount > 0)
{
    for (int i=0; i < candCount; i++)
    {
        /*Get candidate information */
    }
```
Finding Airports by Code

If you want to find a specific airport then use the geocoding type:

```java
USA_CustomGeocoder.GEOCODE_TYPE_AIRPORTS_BY_CODE
```

/* Input address example */
String code = "ALB";
USA_UserInputAddress inpAddr = new USA_UserInputAddress();

/* Set the input Address */
inpAddr.setStreet(code); /* Code 'ALB' */

If you were using the MapMarker Core API, you would use the AddressImpl class.

```java
AddressImpl inpAddr = new AddressImpl();

/* Set the input Address */
String country="USA"
inpAddr.setMainAddress(code); /* code 'ALB' */
inpAddr.setCountry(country); /* country 'USA' */
```

For detailed information on all classes, refer to the API documentation for MapMarker Plus USA. It is located in the SDK\MapMarker-4.0.0\docs folder under the directory where you installed the product.

Use the geocode method of the MMJEngine class as shown in this example. This class is used for geocoding with either the MapMarker Plus USA Java API or the MapMarker Core Java API. To geocode using the MMJClient class see Creating a Web Geocoding Client in Java on page 54.

```java
/* Geocode Address */
MMJEngine engine = new MMJEngine();
ClientGeocodeResponse geoRes = null;
geoRes = engine.geocode(USA_CustomGeocoder.GEOCODE_TYPE_AIRPORTS_BY_CODE,
inpAddr, geoCon);
```

The airport that matches the input code will be returned. Now you can use the USA_UserAirportCandidateAddress to obtain detailed candidate information.

```java
USA_UserAirportCandidateAddress candAddr = new
USA_UserAirportCandidateAddress(geoRes.candidateAt(0));
```
MapMarker Sample Application

The MapMarker sample application is a Java client that illustrates how to geocode U.S. addresses using the MapMarker Plus 11.0 developer product.

In this chapter:

- Requirements and Setup ............................................. 65
- Using the Sample Application ................................. 65
Requirements and Setup

Before you try to geocode an address using the application, make sure that you have access to a MapMarker Plus USA server. To start the server, run the startup.bat (Windows) or startup.sh (UNIX) from the \SDK\MapMarker-4.0.0\bin directory in the path where you installed MapMarker. Windows users may also use the program shortcut in the Start menu.

Be sure that your classpath is modified to include path(s) to the following list of jar files.

- mmjclient.jar
- mmjclient_usa.jar
- micsys.jar
- miutil.jar
- xercesImpl.jar
- jdom.jar
- mmjsample_usa.jar

Using the Sample Application

Run the sample application (USA_MMJSample.class) from the \SDK\MapMarker-4.0.0\bin directory where you installed MapMarker. Windows users may also use the Start menu program shortcut.
Testing the Server

The server must be running properly before you can perform any successful geocoding. To test the server, use the sample application's URL server test area. The URL to the MapMarker Plus USA server is already filled in. You can change the URL to the location of a different server. Click the Test button. If you have successfully connected to a MapMarker Plus USA server, the text under the URL displays Server: VALID. If the connection is not successful, then the text under the URL displays Server: INVALID.

Setting Preferences

There are three preferences you can set that modifies the results of the geocode process.

- Close Match Only – specifies the geocoder return only those candidates that have been flagged as close matches to the input address.
- Fall Back to Postal Code – the geocoder attempts to geocode to the postal centroid if no candidates were returned from an address geocode and a postal code was supplied.
- Fall Back to Geographic – the geocoder attempts to geocode to the urban area centroid or the province centroid, depending on the data available.
- Max Candidates – enables you to change the number of candidates returned from the server.
- Dictionary Options – enables you to specify whether you want candidates to come from address dictionaries, user dictionaries, or both. If using both, you can also give preference to candidates from either address dictionaries or user dictionaries.
- Geocode Type – enables you to specify whether to geocode to street, postal code, or geographic centroid.

Setting Must Match Options

There are also several must match options that you can select to change the number of candidates returned from the server. These options specify that the returned candidates must match certain parts of an address. These include:

- House Num (house number)
- Street
- State
- Zipcode
- City
- CASS

Postal Geocoding

If you provide a postal code only, you can perform a postal geocode. To attempt a postal geocode, provide a valid postal code in the Postal Code text box and click Geocode Address at the bottom of the application.

Area Geocoding

If you provide a city or town name, you can geocode to the city or town. Obviously this is not very accurate and is useful only if none of the other geocoding options are available.
Address Geocoding

If you provide a street address, and either a town and state, or a postal code, you can perform an address geocode. To attempt an address geocode, use the Street Name text box to input the street, including a house number if available. Use the Postal Code text box to input the ZIP Code and/or use the City Name text box to input the town. When geocoding an address you may change the preferences and must match options. To geocode the address, click Geocode Address at the bottom of the application.

Note: MapMarker allows you to geocode to street intersections. The two streets must be separated by an "&&" for MapMarker to recognize them as a street intersection.

Viewing Candidates

If there are any candidates returned from the server they are displayed in the Candidates section of the application. This table includes the following columns of information:

- CloseMatch (True or False)
- Firm
- Street
- City
- State
- Zip
- Zip4

The following information is also displayed in the application.

- Census
- Long (Longitude)
- Lat (Latitude)
- Result

Candidate Statistics

The Candidate Statistics frame displays the number of possible candidates, close match candidates, and total returned candidates.
Deploying Applications

This chapter discusses how to deploy your application as a Web servlet and how you can integrate MapMarker Plus USA with other MapMarker country geocoders.

In this chapter:

- Deploying MapMarker Plus as a Servlet. . . . . . . . . . . . . . . . 69
- Integration with MapMarker for Other Countries. . . . . . . . . 69
Deploying MapMarker Plus as a Servlet

A servlet is a Java program that runs as part of a network service, typically an HTTP server. The most common use for a servlet is to extend a web server by generating web content dynamically. MapMarker Plus extends the Java servlet class which provides all of the necessary functionality for acting as a web server. MapMarker is J2EE compliant making it easy to deploy in most available application server software.

Using the Default Tomcat Servlet Container

MapMarker Plus is installed in the Tomcat Servlet Container 5.5.9. The installer provides this example environment so that you can run the servlet and sample application immediately after finishing the installation. Simply use the default shortcut to start the server.

On Windows platforms the default shortcut is located under the Windows Start menu, Programs > MapMarker Plus > Start Server.

On UNIX platforms the default shortcut is installed in your home directory and is called Start_Server.

Deploying MapMarker Server in Other Web Environments

MapMarker Plus is verified and supported on Sun One 7.0, WebLogic 8.1, and WebSphere 5.1 and 6.0.

For information about deploying servlet applications and for information on servlet container specific requirements, see the documentation that was delivered with your servlet container.

For information about servlet technology and creating a war file, read the documentation on the Sun Web site:


Sun also provides a list of servlet containers on the following Web site, including links to those products:

http://java.sun.com/products/servlet/industry.html

More information about the Tomcat Servlet Container can be found online at:

http://jakarta.apache.org/tomcat/index.html

Integration with MapMarker for Other Countries

If you plan on using multiple countries (e.g., Canada and USA) with MapMarker, you have several integration options. The following options are discussed:

• Running one Tomcat web server for each country.
• Integrating multiple countries in one servlet context.
Running One Tomcat Web Server for Each Country

The default HTTP port for MapMarker Plus is 8095. You may use the default port for each country you install. If this is the case, the same URL is used for all Tomcat web servers. Therefore, only one Tomcat (i.e., country) web server may run at a time, since each Tomcat web server would be using the same port. To allow multiple Tomcat web servers to run simultaneously, you must specify a unique installation directory and unique HTTP startup and shutdown ports for each Tomcat web server. You can specify the installation directory and the HTTP startup and shutdown ports during installation.

To alter the ports after installation, modify server.xml in $TOMCAT_HOME/conf. This scenario allows unique URLs for each Tomcat web server. Note that running multiple web servers simultaneously may decrease performance.

Integrating Multiple Countries in One Servlet Context

In order to allow multiple countries to run under one URL, you need to do the following:

1. Install the first country. Expand the mapmarker war file created by the installer into the proper location for servlet contexts. Consult the Application Server's documentation for the proper servlet context location on the file system.
2. Install the subsequent countries. You must install, as a minimum, the MapMarker server component and country data.
3. For each country you wish to integrate into the existing servlet context, copy the country specific jar files and properties files from the server installation in the previous step into the servlet context for the first country.
   Country specific jar files are found under the lib directory in the servlet context; country specific property files can be found under the classes directory. Ensure the directory structure is maintained under the classes directory when copying files to a new location. If it is necessary to move the country data to a different location, you must update the property file path information located in the classes directory.
4. Verify settings (such as MapMarker data location, and library location) in the country specific properties files are appropriate for the given Application Server.
5. Restart the Application Server.

Note: MapMarker Plus 11.0 is built upon the MapMarker Java core version 4.0. Older core versions (2.0 and 3.0) cannot be integrated into one servlet context with MapMarker Plus 11.0.
Creating and Using Custom Address Dictionaries

This chapter describes how to create a user dictionary programmatically, and explains how you can configure and set preferences for your dictionaries using the MapMarker API.

In this chapter:

- Creating a Custom Dictionary ........................................... 72
- Using Custom Dictionaries ............................................... 76
Creating a Custom Dictionary

We recommend that you use the User Dictionary Wizard in the MapMarker Plus desktop product to create a user dictionary.

Custom user dictionaries created in MapMarker Plus v. 10.x and earlier are not compatible with MapMarker 11.0. We recommend that you re-create the dictionary in MapMarker Plus 11.0 with the User Dictionary Wizard using the original source data. If this is not feasible, we have provided a command-line utility that converts an existing custom user dictionary to a dictionary that uses the MapMarker Plus 11.0 format.

For more information on the User Dictionary Wizard and converting existing dictionaries to the MapMarker Plus 11.0 format, see Chapter 8: User Dictionaries in the MapMarker Plus Desktop User Guide.

To create a user dictionary programmatically with MapMarker Plus 11.0, you must use the C-based Data Dictionary API with the JNI adapter. The adapter will translate the calls to the MapMarker Java API so that the dictionary can be created in MapMarker Plus 11.0.

The remainder of this section discusses the required and optional fields for user dictionaries, and describes the Data Dictionary API.

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 Abbrevation</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>
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

When creating a user dictionary, you must specify a base name of eight characters or less. Each dictionary resides in its own directory, and each one 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. Some of the output files are tied to the base name. The other output files have constant names. For example, the output files for a dictionary called ud1 are the following:

- postinfo.jdr
- postinfo.jdx
- lastline.jdr
- post2sac.mmj
- geo2sac.mmj
- sac2fn_ud.mmj
- ud1.jdr
- ud1.jdx
- ud1.bdx

* These fields are highly recommended.
If your data includes placenames, the dictionary contains the following files:

```
ud1.pdx
ud1.pbx
```

The dictionary also contains these log files:

```
ud1.log
ud1.err
```

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.dll` 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>
</table>
| geoEngHandle          | An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to `GeoEngInit()`.
| input_table_name      | Full path of the MapInfo input table name from which a user dictionary is created.                                                          |
| output_dictionary_name| Full path to the output dictionary including the user dictionary file extension `.udr`.                                                    |
| pUDInfo               | InfoPointer to the UD_Info (user dictionary information) structure                                                                         |
| pStatusFunction       | Pointer to status function                                                                                                               |

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 postinfor and post2sac files and to look up city names in the `geo2sac` 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;
```
Using Custom Dictionaries

The MapMarker Java API enables you to configure and set preferences for the custom dictionaries that you have created.

To configure a user dictionary, you must specify a path to it. This is done in the USA_DataManagerSettings.properties file on the server.

Preferences can be set for a dictionary either from the server, or in a geocode request at runtime.

Configuring User Dictionaries on the Server

To configure a user dictionary on the server, you must add the path to the dictionary as an entry to the USA_DataManagerSettings.properties file. The properties file resides on the server machine and lists all the configured dictionaries. The engine accesses the properties file to find the locations of the dictionaries.

The path to each user dictionary is added to the file as a dictionary path key, as shown in the code example below.

```java
/** Optional - The number of dictionaries to be loaded. DEFAULT=1 */
DICTIONARY_COUNT=2

/** Required - The path to the highest ranking dictionary.
 * Note that DICTIONARY_PATH is required from 1 to DICTIONARY_COUNT.
 */
DICTIONARY_PATH1=H:\mmj\data\USA\address_dictionary
DICTIONARY_PATH2=H:\mmj\data\USA\user_dictionary
```

The dictionary path key includes the location of the dictionary and its rank. The integer at the end of the key represents the rank. The rank indicates the order in which configured dictionaries are searched and can be an integer from 1 to N. The highest ranking dictionary has a rank of 1. The number of configured dictionaries specified in the properties file must match the value of `DICTIONARY_COUNT`.

All user dictionaries specified in the USA_DataManagerSettings.properties file are considered when you are geocoding or browsing records, depending on the `DICTIONARY_COUNT` value and any client preferences that the user has set.

User Dictionary Preferences

MapMarker 11.0 supports the use of multiple dictionaries in a geocoding operation. Additional preferences allow you specify which available dictionaries to use in a geocoding operation. These preferences can be set either directly on the server, or in a geocoding request using the MapMarker Java API.

Specifying Preferences on the Server

To indicate which dictionaries to use, edit the configured dictionaries in the USA_DataManagerSettings.properties file. Add path keys for the dictionaries you want to use, and remove path keys to dictionaries that you do not want to use. Make sure that you update the `DICTIONARY_COUNT` to reflect the correct number of dictionaries.
### Specifying Preferences Using the MapMarker Java API

Users of a server product can specify which dictionaries to use in a geocode request. These preferences can be set using the IDictionarySearchOrder object. To retrieve the object, use the getDictionarySearchOrder method.

The IDictionarySearchOrder object retrieves a list of the configured dictionaries and contains information about each one. It returns the count of the number of configured dictionaries, plus the following information for each dictionary:

- configured index
- description string
- if dictionary is a user dictionary
- whether user searches this dictionary by default

Once you have the IDictionarySearchOrder object, you can modify it using a number of methods to set the dictionary preferences for the geocode operation and then add the object to the GeocodeConstraints to be passed in with the geocode request.

Some of these method is described briefly in the following table:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDictionaryAvailableForSearch</td>
<td>Returns true if dictionary will be searched.</td>
</tr>
<tr>
<td>setDictionaryAvailableForSearch</td>
<td>Setting to false indicates that the dictionary will not be searched.</td>
</tr>
<tr>
<td>getDictionaryDescription</td>
<td>Returns the description of the indicated dictionary.</td>
</tr>
<tr>
<td></td>
<td>Descriptions are located in a plain text file named “dictionarydesc.txt” in</td>
</tr>
<tr>
<td></td>
<td>the dictionary path.</td>
</tr>
<tr>
<td></td>
<td>This description is created when the user dictionary is created. If no file</td>
</tr>
<tr>
<td></td>
<td>is found, a string indicating whether the dictionary is a user dictionary will</td>
</tr>
<tr>
<td></td>
<td>be returned.</td>
</tr>
</tbody>
</table>

Dictionaries may be removed from the list of dictionaries to search by using a setDictionaryAvailableForSearch method on the IDictionarySearchOrder object, and then adding it to the constraints for the geocode request.

To find out which dictionary a candidate came from, you can use the getDictionaryNumber method in the CandidateAddress class. You can use this number to get a description of the configured dictionary in the dictionarydesc.txt file for the dictionary. If this file is missing, the description of the dictionary will be limited to “Address Dictionary” or “User Dictionary.”

To pass in the IDictionarySearchOrder object with the geocode constraints, use the setDictionarySearchOrder method in the IGeocodeConstraints interface. It sets a custom order for searching configured user dictionaries.

For information on specific properties and methods, please see the javadocs. Browse to http://localhost:8095 on a computer where the MapMarker Server is running and click on "MapMarker Plus USA Java API" in the box labeled MapMarker Documentation.
Client Preferences

In a client application, users set preferences that give priority to the user dictionaries if the MapMarker Address dictionary is also used, and enable the user to specify whether to use the Address dictionary, user dictionaries, or both.

The following table describes the client preferences:

<table>
<thead>
<tr>
<th>Preference Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFER_UD</td>
<td>Use both address dictionaries (AD) and user dictionaries (UD) but give preference to candidates from UD.</td>
</tr>
<tr>
<td>PREFER_AD</td>
<td>Use both address dictionaries (AD) and user dictionaries (UD) but give preference to candidates from AD.</td>
</tr>
<tr>
<td>UD_ONLY</td>
<td>Use only user dictionaries (UD). Ignore address dictionaries (AD).</td>
</tr>
<tr>
<td>AD_ONLY</td>
<td>Use only address dictionaries (AD). Ignore user dictionaries (UD).</td>
</tr>
<tr>
<td>AD_AND_UD</td>
<td>Use both address dictionaries (AD) and user dictionaries (UD) with no special preference to candidates. This is the default.</td>
</tr>
</tbody>
</table>
Converting MapMarker J Server Applications

This appendix gives a code example of converting a MapMarker J Server 3.1 application to MapMarker Plus 11.0.

In this appendix:

* J Server 3.1 to MapMarker Java 11.0 Conversion . . . . . . . . . . . . . . . . . . . . 80
J Server 3.1 to MapMarker Java 11.0 Conversion

This appendix provides an example of converting a MapMarker J Server 3.1 application to MapMarker Java 11.0.

Text Conventions

The following text conventions have been used in this example to illustrate what has been added for MapMarker Plus 11.0, and what has been deprecated.

- Bold text indicates changes or additions for MapMarker Plus 11.0.
- Underlined text indicates deprecated classes.

For a full list of deprecated classes for the MapMarker Java API, see the javadocs. Browse to http://localhost:8095 on a computer where the MapMarker Server is running and click on "MapMarker Plus USA Java API" in the box labeled MapMarker Documentation.

Code Example

```java
import com.mapinfo.mapmarker.USA.USA_GeocodeConstraints;
import com.mapinfo.mapmarker.USA.USA_UserCandidateAddress;
import com.mapinfo.mapmarker.USA.USA_UserInputAddress;

// remove deprecated client imports packages*/
import com.mapinfo.mapmarker.client.ClientGeocodeResponse;
import com.mapinfo.mapmarker.client.MMJClient;

// add new MM 11 user packages */
import com.mapinfo.mapmarker.user.ClientGeocodeResponse;
import com.mapinfo.mapmarker.user.MMJClient;
import com.mapinfo.mapmarker.user.MapMarkerFatalException;
import com.mapinfo.mapmarker.user.MapMarkerException;

public class USA_JServerSample {
    private void geocodeSampleAddress() {
        /* Set server url*/
        String url = "http://199.29.2.75:8096/mapmarker40/servlet/mapmarker";

        /* Input address example */
        String firm = "MapInfo";
        String addr = "1 Global view";
        String city = "Troy";
        String state = "NY";
        String zip5 = "12180";
        String zip4 = "";

        USA_GeocodeConstraints geoCon = new USA_GeocodeConstraints();

        /* setProperty no longer defined */
        geoCon.setProperty(USA_GeocodeConstraints.KEY_MAXCANDIDATES,"1");
        geoCon.setProperty(USA_GeocodeConstraints.KEY_MUST_MATCH_MAINADDR,"true");

        /* MM 11.0 changes */
    }
}
```
geoCon.setMaxCandidates(1);
geoCon.setMustMatchMainAddress(true);

USA_UserInputAddress inpAddr = new USA_UserInputAddress();

/* Set the input Address */
inpAddr.setFirm(firm);    /* Firm: MapInfo Corp */
inpAddr.setStreet(addr); /* Street: 1 Global View */
inpAddr.setCity(city);    /* City: Troy */
inpAddr.setState(state);  /* State: NY */
inpAddr.setZipcode(zip5); /* ZipCode: 12180 */
inpAddr.setZip4(zip4);    /* Zip+4: 8399 */

System.out.println("Input Address:" + inpAddr.toString());
System.out.println();
System.out.println("GeoConstraints:" + geoCon.toString());

ClientGeocodeResponse geoRes = null;

/* MM 11, added error handling with try\catch statements */
try
{
    geoRes = MMJClient.geocode(MMJClient.GEOCODE_TYPE_ADDRESS, inpAddr,
           geoCon, url);

    /* check response code no longer valid, must use exception handling*/
    if (geoRes.getResponseCode() == 0) {

        /* Get candidate count */
        int candCount = geoRes.candidateCount();

        /* Get returned candidates */
        if (candCount > 0)
        {
            /* Get candidate information
            * Review Java docs for a complete list of candidate information */
            for (int i = 0; i < candCount; i++)
            {
                /* changed method calls */
                USA_UserCandidateAddress candAddr = new USA_UserCandidateAddress();
                candAddr.copy(geoRes.candidateAt(i));

                /* MM 11.0 changes */
                USA_UserCandidateAddress candAddr  = new USA_UserCandidateAddress(geoRes.candidateAt(i));

                System.out.println("Candidate #" + i + candAddr.toString());
                System.out.println("Close Match:" + candAddr.isCloseMatch());
                System.out.println("Firm:" + candAddr.getFirm());

                /* changed getFormattedStreet property */
                System.out.println("Street:" +
                                   candAddr.getFormattedStreet());
            }
        }
    }
}

/* MM 11 changes*/
System.out.println("Street:" +
candAddr.getFormattedStreetAddress());

System.out.println("City:" + candAddr.getCity());
System.out.println("State:" + candAddr.getState());
System.out.println("Zip:" + candAddr.getZipcode());
System.out.println("Zip4:" + candAddr.getZip4());
System.out.println("X:" +
String.valueOf(candAddr.getLocation().x));
System.out.println("Y:" +
String.valueOf(candAddr.getLocation().y));

/* changed getResultCode Property*/
System.out.println("ResultCode:" + candAddr.getResultCode());

/* MM 11 changes*/
System.out.println("Street:" + candAddr.getGeoResult());
System.out.println("Census:" + candAddr.getCensusBlock());

} // end for
} // end if, check candidate count

} // end try
/* MM 11 exception handling */
catch (MapMarkerFatalException e)
{
    System.out.println("Fatal Exception:" + e.getMessage());
}
catch (MapMarkerException e)
{
    System.out.println("Exception:" + e.getMessage());
}

/* Response code and message no longer valid */
else
{
    System.out.println("Response Code:" + geoRes.getResponseCode());
    System.out.println("Response Message:" +
    geoRes.getResponseMessage());
}
} /* End */

/**
 * The main method will only be called when this test is run stand alone.
 */
public static void main(String[] args) {
    USA_JServerSample geocodeTest = new USA_JServerSample();
geocodeTest.geocodeSampleAddress();
}
JNI Adapter Configuration

File Settings

This appendix explains the settings you can modify in the configuration file of the JNI adapter.

In this appendix:

- JVM Settings .................................................. 84
- Country and Coordinate System Settings ............... 84
JVM Settings

The configuration file controls the creation of the JVM in which the Java engine runs. The JVM settings are useful for changing the amount of memory used by the JVM.

Here are the settings:

+numVmOptions=<the number of settings for the JVM>
-<Djava.class.path>=<the series of jar files required to geocode addresses in the selected country>
-server-causes a server JVM to be created (Windows only)
-Xms128m-minimum heap size for JVM to create when initialized (128 M)
-Xmx256m-maximum size JVM heap can grow to (256 M)
-Xmn40m-another memory related setting for the JVM

In this example, the +numVmOptions setting would be set to 5.

The following options are used to create the JVM but are not strictly JVM settings:

+jvmVersion=0x00010004-the version of JVM to create, in this case 1.4
+jvmPath-the full file path to the jvm.dll (including the filename jvm.dll) (Windows only)

Country and Coordinate System Settings

The following setting indicates the country that the adapter is going to use for geocoding.

+country=<the country this configuration file is for geocoding>

All classes for this country must be given in the -Djava.class.path setting in the JVM settings described in the previous section.

The following settings control the coordinate system. These settings indicate the standard and alternate coordinate systems of the returned coordinates. The examples show the U.S. settings of NAD83 as the standard coordinate system and NAD27 as the alternate.

+clientCRS=epsg:4269-the standard coordinate system to return coordinates in, operates as NAD83 in the USA product (this string is the string for NAD83).

+altClientCRS=epsg:4267-the alternate coordinate system to return coordinates in, the result when calling GeoEngNAD83ToNAD27 (this string is the string for NAD27).

The standard coordinate system takes the place of NAD83 in GeoEngNad83ToNAD27, and the alternate coordinate system takes the place of NAD27.
This appendix 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)

In this appendix:

- MapMarker Plus 11.0 Notes on OLE Automation ........ 86
- Running MapMarker Server on Windows Systems ........ 86
- Creating a Custom Geocoding Control ................. 88
- Geocoder Control Properties, Events, and Methods .... 88
- OLE Automation Methods .................................. 92
- OLE Automation Objects ................................. 125
- Programming Usage Example ............................ 127
MapMarker Plus 11.0 Notes on OLE Automation

To create a custom geocoding control, you must use the OLE Automation API. In previous versions of MapMarker Plus. For MapMarker Plus 11.0, calls are passed either directly through the JNI adapter to the MapMarker Plus Java API, or to the RPC Server, and then through the JNI adapter to the MapMarker Plus Java API.

**Note:** This API is deprecated, and will no longer be supported when MapMarker Plus 12.0 is released.

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 **CONTROL PANEL > ADMINISTRATIVE TOOLS** in Windows 2000, 2003, 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 Start 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 Start 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 2000/2003/XP 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., 11.0, even if you are running one of the point releases of that version, e.g., 11.1, 11.2.
If you encounter a problem with the Name Service Provider that prevents MapMarker Server from running when your machine boots, modify the Windows Services as follows:

1. In the Windows Control Panel, double-click on **NETWORK CONNECTIONS**.
2. Double-click on **LOCAL AREA CONNECTION** to display the **LOCAL AREA CONNECTION STATUS** dialog.
3. In the General tab, click Properties. The Local Area Connection Properties dialog displays.
4. Click Client for Microsoft Networks to highlight it, and then click Properties.
5. In the RPC Service tab, change the Name service provider to DCE Cell Directory Service.
6. In the **NETWORK ADDRESS** box, enter the machine’s IP address. If the machine is not on a network, leave this box blank.
7. Click OK.

To stop the MapMarker service:

1. Choose **CONTROL PANEL > ADMINISTRATIVE TOOLS > SERVICES** and highlight MapMarker Server.
2. Click **STOP**.

To remove the MapMarker service from Windows Services:

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

**Console Application**

To start MapMarker Server as a console application on Windows 2000/2003/XP:

- 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 2000/2003/XP 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., 11.0, even if you are running one of the point releases of that version, e.g., 11.1, 11.2.

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.
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.

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.

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 geogeoengine mm32v_.dll. Default: False.</td>
</tr>
<tr>
<td>BClosetCandidates</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>
<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>
</tbody>
</table>
### Input Properties

<table>
<thead>
<tr>
<th>Input Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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: FEET 0, DEGREES 1, 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>
<tr>
<td>Precision</td>
<td>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</td>
</tr>
<tr>
<td>NumCandidates</td>
<td>Integer: The number of match candidates for the record.</td>
</tr>
</tbody>
</table>
### Output Properties

<table>
<thead>
<tr>
<th>Output Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumCloseCandidates</td>
<td>Integer: The number of close match candidates for the record.</td>
</tr>
<tr>
<td>LastErrorCode</td>
<td>Long: Returns the last error code generated.</td>
</tr>
<tr>
<td>CensusBlockID</td>
<td>String: Census Block tabulation number.</td>
</tr>
<tr>
<td>ResultCode</td>
<td>String: Geocoding result code.</td>
</tr>
<tr>
<td>DatabaseTypes</td>
<td>Integer: a number that identifies the available databases:</td>
</tr>
<tr>
<td></td>
<td>2 - ZIP Centroid database</td>
</tr>
<tr>
<td></td>
<td>3 - Street and ZIP Code databases</td>
</tr>
<tr>
<td></td>
<td>4 - User Dictionary</td>
</tr>
<tr>
<td></td>
<td>6 - User Dictionary and ZIP Code databases</td>
</tr>
<tr>
<td></td>
<td>7 - Street, User Dictionary, and ZIP Code databases</td>
</tr>
<tr>
<td>StringBinding</td>
<td>String: RPC binding string used to connect to the server.</td>
</tr>
<tr>
<td>PrimaryStreet</td>
<td>String: The candidate primary street, if it has one.</td>
</tr>
<tr>
<td>RecordType</td>
<td>String: The candidate ZIP + 4 record type as categorized by the U.S. Postal</td>
</tr>
<tr>
<td></td>
<td>Service. If the match is from a TIGER record, the candidate may not have</td>
</tr>
<tr>
<td></td>
<td>a record type.</td>
</tr>
<tr>
<td></td>
<td>F – Firm</td>
</tr>
<tr>
<td></td>
<td>G – General Delivery</td>
</tr>
<tr>
<td></td>
<td>H – Highrise</td>
</tr>
<tr>
<td></td>
<td>P – PO Box</td>
</tr>
<tr>
<td></td>
<td>R – Rural Route/Highway</td>
</tr>
<tr>
<td></td>
<td>S – Street</td>
</tr>
<tr>
<td>DeliveryPoint</td>
<td>String: The candidate delivery point.</td>
</tr>
<tr>
<td>Carrier Route</td>
<td>String: The candidate carrier route.</td>
</tr>
<tr>
<td>CheckDigit</td>
<td>String: The candidate check digit.</td>
</tr>
<tr>
<td>TabbedAddress</td>
<td>String: The candidate street address components delimited by tabs as follows:</td>
</tr>
<tr>
<td></td>
<td>House Number</td>
</tr>
<tr>
<td></td>
<td>Directional Prefix</td>
</tr>
<tr>
<td></td>
<td>Street Type Prefix</td>
</tr>
<tr>
<td></td>
<td>Street Name</td>
</tr>
<tr>
<td></td>
<td>Street Type Suffix</td>
</tr>
<tr>
<td></td>
<td>Directional Suffix</td>
</tr>
<tr>
<td></td>
<td>Unit Type</td>
</tr>
<tr>
<td></td>
<td>Unit Value</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>be converted from a rural route address to a city-style address by</td>
</tr>
<tr>
<td></td>
<td>using a USPS product, known as the Locatable Address Conversion Service.</td>
</tr>
</tbody>
</table>
OLE Automation Methods

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

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

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

- **DoGeocode().** 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.
Appendix C: OLE Automation API

- **DoSetProperties()**
  Brings up a set of property pages for the Geocoder Control.
- **DpvGeocodeAddress()**
  Turns DPV mode on, and then geocodes an address and builds a list of candidates.
- **DpvGeocodeAddressLastLine()**
  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()**
  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()**
  Attempts to geocode an address and build a list of candidates.
- **GeocodeAddressEx()**
  Attempts to geocode an address and build a list of candidates. Also enables street2 information to be passed in as a separate field.
- **GeocodeAddressLastLine()**
  Enables you to enter an input address with an unparsed line containing the city, state, and/or ZIP Code.
- **GeocodeAddressLastlineEx()**
  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()**
  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()**
  Requires user to input a serial number for geocoding.
- **GeocodeAddressWithSerialEx()**
  Requires user to input a serial number for geocoding, and enables street2 information to be passed in as a separate field.
- **GeocodeCheckDbAvailability()**
  Checks to see which types of database(s) are available for geocoding.
- **GeocodeFreeSet()**
  Frees the server side information after geocoding.
- **GeocodeGetCandidates()**
  Gets all the candidate information after geocoding.
- **GeocodeGetErrorText()**
  Returns a short text description of the specified error.
- **GeocodeGetHwyExitCandidate()**
  Gets the specified highway exit candidate after a call to GeocodeHwyExit_A().
- **GeocodeGetServerVersion()**
  Returns the server version number.
- **GeocodeGetStatesFound()**
  Returns a list of 2-digit state abbreviations representing states found in the Address Dictionary path. These are not necessarily licensed states.
- **GeocodeGetStatesLicensed()**
  Returns a list of 2-digit state abbreviations representing licensed states.
- **GeocodeHwyExit()**
  Returns a count of the number of highway records based on input criteria of highway exit and state.
• GeocodeIsCandidateMultiUnit(). ............................... 117
  Returns a True/False value, indicating whether the candidate address contains multiple units.
• GeocodePostalCentroid() .................................. 117
  Geocodes a ZIP Code or ZIP + 4 centroid.
• GeocodePostalCentroidWithSerial() ......................... 118
  Requires a serial number for ZIP Code-level geocoding.
• GetCandidateAt() ............................................ 119
  Returns the line of text in the Match Candidate list box.
• GetCandidateCensusBlockIDAt() .............................. 119
  Returns the Census Block ID as a string.
• GetCandidateCityAt() ...................................... 120
  Returns the city portion of the address as a string.
• GetCandidateFirmAt() ...................................... 120
  Returns the firm portion of the address as a string.
• GetCandidateLatitudeAt() .................................. 120
  Returns the latitude as a double.
• GetCandidateLongitudeAt() ................................ 121
  Returns the longitude as a double.
• GetCandidatePlus4At() ..................................... 121
  Returns the ZIP add on of the address as a string.
• GetCandidatePrecisionAt() ................................ 122
  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() ................................ 122
  Returns the result code portion of the address as a string.
• GetCandidateStateAt() ...................................... 122
  Returns the state portion of the address as a string.
• GetCandidateStreetAt() .................................... 123
  Returns the street portion of the address as a string.
• GetCandidateZIPAt() ........................................ 123
  Returns the ZIP portion of the address as a string.
• GetFullName() .................................................. 124
  Returns the full name and path of the application.
• GetName() ...................................................... 124
  Returns the application name.
• GetVersionNum() .............................................. 124
  Returns the version number of the application.
• RefreshDialog() .............................................. 124
  Forces the dialog box to repaint.
• SelectCandidateAt() ......................................... 125
  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
ClearDialogText()

Returns
The LastErrorCode property can be used to determine the nature of any errors that occur.

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
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 Map-</td>
</tr>
<tr>
<td></td>
<td>Marker Server is running. NULL or an empty string assumes MapMarker Server</td>
</tr>
<tr>
<td></td>
<td>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.

<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>
</tbody>
</table>
Disconnect()

Purpose
This call is used to disconnect from the MapMarker Server.

Syntax
    Disconnect() 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.

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.
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.

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>
</tbody>
</table>
Returns
TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur.

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.

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

<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>
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>
<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] The total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close candidates.</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.

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.
Syntax

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</td>
</tr>
<tr>
<td></td>
<td>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>status</td>
<td>[Output Integer] Lastline information (city, state, ZIP).</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The total number of candidates.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Output Integer] The number of close match candidates.</td>
</tr>
<tr>
<td></td>
<td>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.
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.

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>
</tbody>
</table>
Returns TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur.

GeocodeAddress()

Purpose

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

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>
<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>

Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur.

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 127, for a complete sample program that illustrates this function.

```vbnet
Private Sub cmdGeocode_Click()
    Dim retVal As Boolean
    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.

**Syntax**

```vbscript
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>status</td>
<td>[Output Integer] The match status of the geocode. See GeocodeAddress() on page 102 for a full description of the status parameter.</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.

## 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).

### Syntax

```
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>
</tbody>
</table>
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur.

### 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 – 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] The total number of candidates found.</td>
</tr>
<tr>
<td>numCloseCandidates</td>
<td>[Output Integer] The number of close candidates.</td>
</tr>
</tbody>
</table>
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.

**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.
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.

**GeocodeAddressWithSerial()**

**Purpose**

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

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 GeocodeAddressWithSeral(); however, it also enables you to pass in street2 information in a separate field.

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.

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.

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
GeocodeFreeSet(geocodeHandle As Long) As Boolean

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| geocodeHandle  | [Input/Output] Used to reference information about the geocoding opera-
|                | tion in other calls.                        |
Returns

TRUE if successful, FALSE if not. The LastErrorCode property can be used to determine the nature of any errors that occur.

GeocodeGetCandidates()

Purpose

This call returns all the needed candidate information from the GeocodeAddress() call. This information is stored in a collection class.

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.

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 127, for a complete sample program that illustrates this function.

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
End Sub
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.

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.

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.

GeocodeGetServerVersion()

Purpose

This method returns the version of the server.

Syntax

```
GeocodeGetServerVersion(pVersionNum As Double) As Boolean
```
GeocodeGetStatesFound()

Purpose
This method returns a list of states in the Address Dictionary path.

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.

GeocodeGetStatesLicensed()

Purpose
This method returns a list of licensed states.

Syntax
GeocodeGetStatesLicensed(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.
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>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 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.
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.

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>
</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.

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).

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.</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.

### GetCandidateAt()

**Purpose**

This call returns the text associated with a specific candidate in the Match Candidates list box.

**Syntax**

```vbscript
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.

### 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.

**Syntax**

```vbscript
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.
GetCandidateCityAt()

**Purpose**
This call returns the city portion of an address in the Match Candidates list box.

**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.

GetCandidateFirmAt()

**Purpose**
This call returns the firm portion of an address to the Match Candidates list box.

**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.

GetCandidateLatitudeAt()

**Purpose**
This call returns the latitude of an address in the Match Candidates list box.

**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.

GetCandidateLongitudeAt()

Purpose

This call returns the longitude of an address in the Match Candidates list box.

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.

GetCandidatePlus4At()

Purpose

This call returns the ZIP Add-on portion of an address in the Match Candidates list box.

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.
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).

**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.

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.)

**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.

GetCandidateStateAt()

**Purpose**
This call returns the state portion of an address in the Match Candidates list box.

**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.

GetCandidateStreetAt()

Purpose

This call returns the street portion of an address in the Match Candidates list box.

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.

GetCandidateZIPAt()

Purpose

This call returns the ZIP Code portion of an address in the Match Candidates list box.

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.
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.

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.

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.

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.
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.

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 value of DpvConfCode determines the value of DpvCmra and DpvFalsePos. 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). The DpvCmra can be one of the following:
  Y–A CMRA was found.
  N–A CMRA was not found.
  blank–If the DpvConfCode returns an 'N', the DpvCmra property will be blank. Since the address does not exist in the DPV database, there is no reason to check for a 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. The return values can be one of the following:
  Y–DPV has detected that an artificially generated address was geocoded. DPV will shut down immediately.
  N–The address is not an artificially generated address.
  blank–If the DpvConfCode returns a 'Y', 'S', or 'D', the DpvFalsePos property will be blank. Since the address exists in the DPV database, there is no reason to check for a false positive.
• 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.

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 and PMB designator present (PMB123 or #123).
R1–Input address matched to CMRA but PMB designator not present (PMB 123 or #123).

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 'MapMarkr
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
    End If
End Sub
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 "
    End If
End Sub
Case 2
    lblPrec = "Zip+2 
Case 1
    lblPrec = "Zip centroid 
End Select
End If
' 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
            cmdDisconnect.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
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
The code provided is part of an application that interacts with a server using the OLE Automation API. The code snippet demonstrates handling of various UI elements such as enable or disable states for buttons like 'Get Candidates', 'Connect', 'Disconnect', and 'Free Set'. It also shows how to manage geocoding functionality with a few conditional checks and error handling. The logic includes checking for changes in geocoding configuration parameters and handling errors appropriately.
lblStatus = "Server : " & txtServer & " is not running"
End If
Else
lblStatus = "Error geocoding, error code: " & Str$(errorCode)
End If
Else
DatabaseTypes = objMM.DatabaseTypes
Select Case status
  Case 0
    lblStatus = "Single match"
  Case 1
    lblStatus = "Multiple match"
  Case 2
    lblStatus = "No close matches"
  Case 3
    lblStatus = "No candidates"
  Case 4
    lblStatus = "Single intersection match"
  Case 5
    lblStatus = "Multiple intersection match"
  Case 6
    lblStatus = "No close intersection matches"
  Case 7
    lblStatus = "No intersection candidates"
  Case 8
    lblStatus = "Possible intersections"
Case Else
  lblStatus = Str$(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
    & ", " & .Zip & "-" & .plus4
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
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
Index

A

address dictionary
  definition 39
  installing 14
  storing data 16
  storing on hard drive 16
address matching
  geocoding 8
addresses
  browsing 58
  input 42–43
  result code information 46
airport geocoding 62
ASP API 29

B

browsing addresses 58

C

CAddressList 125
candidates
  definition 38
  retrieving 53
city centroid geocoding 59–60
ClearDialogText() 95
close match
  definition 39
Cold Fusion API 29
compiling existing applications 32–34
configuration file
  JNI adapter 26, 84
  Connect() 95–96
console mode installation 23
constraints 50–52
coordinate systems
  JNI adapter configuration file settings 84
  setting in applications 52
country settings 84
creating
  user dictionaries 72–75
  Web client 54–58

d

data
  installing 14
  storing 16
data dictionary API 74–75
database development 31
database drivers 25
deployment as servlet 69
dictionary result information 46
Disconnect() 96
documentation set 10–11
DoGeocode() 96
DoSetProperties() 96
DPV (Delivery Point Validation)
  installation 14
  DpvGeocodeAddress() 97–98
  DpvGeocodeAddressLastLine() 98–99
  DpvGeocodeAddressLastLineWithSerial() 99–100
  DpvGeocodeAddressWithSerial() 101

E

Envinsa 38
ESP SQL Server 31

G

generic preferences 43
geocode requests 45
geocode types 41, 45
GeocodeAddress() 102, 104
GeocodeAddressEx() 104–105
GeocodeAddressLastLine() 105–106
GeocodeAddressLastLineEx() 106–107
GeocodeAddressLastLineWithSerial() 107
GeocodeAddressWithSerial() 108, 110
GeocodeAddressWithSerialEx() 110
GeocodeCheckDbAvailability() 111
GeocodeFreeSet() 111
GeocodeGetCandidates() 112–113
GeocodeGetErrorText() 113
GeocodeGetHwyExitCandidate() 113
GeocodeGetServerVersion() 114
GeocodeGetStatesFound() 115
GeocodeGetStatesLicensed() 115
GeocodeHwyExit() 116
GeocodeIsCandidateMultiUnit() 117
GeocodePostalCentroid() 117–118
GeocodePostalCentroidWithSerial() 118–119
geocoding
  airports 62
  city centroids 59–60
  geocode method 52
  highway exits 61
  model 39–40
  preferences 39
  result codes 45–46
geocoding applications
  choosing development tool 36–38
Geocoding Cartridge 31
geocoding constraints 50–52
geocoding terms 38–39
GeoEng API 29–30
GeoEngDDCreateUserDictionary() 74–75
GetCandidateAt() 119
GetCandidateCensusBlockIDAt() 119
GetCandidateCityAt() 120
GetCandidateFirmAt() 120
GetCandidate LatitudeAt() 120
GetCandidateLongitudeAt() 121
GetCandidatePlus4At() 121
GetCandidatePrecisionAt() 122
GetCandidateResultCodeAt() 122
GetCandidateStateAt() 122
GetCandidateStreetAt() 123
GetCandidateZIPAt() 123
GetFullName() 124
GetName() 124
GetVersion() 124
H
highway exit geocoding 61
HP-UX applications
  JNI adapter 34
  migration path 30
L
Informix DataBlades 31
input addresses
  building 42–43
  setting 49
installation
  console mode 23
  re-installing 24–25
  system recommendations 16
  system requirements 15–16
  UNIX/Linux procedure 23
  Windows procedure 17–23
integrating multiple country geocoders 69–70
J
J Server
  2.x API 28
  3.x API 28–29
  converting application to MapMarker Plus 11.0 80–82
  with MapMarker Plus 10.x 38
Java geocoding applications
  migration path 28
Java Virtual Machine 15
JNI adapter
  configuration file settings 26, 84
  running existing applications 32–34
JVM settings
  JNI adapter configuration file 84
L
Linux support 30
M
MapBasic geocoding API 29
MapMarker J Server
  2.x API 28
  3.x API 28–29
  converting application to MapMarker Plus 11.0 80–82
  with MapMarker Plus 10.x 38
MapMarker Plus 10.x
  support 31, 37
  with J Server 38
MapMarker Plus 11.0
  documentation set 10–11
  sample application 65–66
  transition to 31
  what’s new 10
MapMarker server
  request time outs 88
  running 86
MapMarker Streets
  using with MapMarker 12
MapXtreme 38
matching addresses
  with geographic data 8
migrating existing applications 28–30
multiple country integration 69–70
N
new features and enhancements 10
OLE automation API
- creating a custom control 88
- deprecation status 29
- flow diagram 88
- list of OCX methods 92, 94
- programming example 127–136
- summary of properties, events, methods 88

OLE automation methods
- ClearDialogText() 95
- Connect() 95–96
- Disconnect() 96
- DoGeocode() 96
- DoSetProperties() 96
- DpvGeocodeAddress() 97–98
- DpvGeocodeAddressLastLine() 98–99
- DpvGeocodeAddressLastLineWithSerial() 99–100
- DpvGeocodeAddressWithSerial() 101
- GeocodeAddress() 102, 104
- GeocodeAddressEx() 104–105
- GeocodeAddressLastLine() 105–106
- GeocodeAddressLastLineEx() 106–107
- GeocodeAddressLastLineWithSerial() 107
- GeocodeAddressWithSerial() 108, 110
- GeocodeAddressWithSerialEx() 110
- GeocodeCheckDbAvailability() 111
- GeocodeFreeSet() 111
- GeocodeGetCandidates 112–113
- GeocodeGetErrorText() 113
- GeocodeGetHwyExitCandidate() 113
- GeocodeGetServerVersion() 114
- GeocodeGetStatesFound() 115
- GeocodeGetStatesLicensed() 115
- GeocodeHwyExit() 116
- GeocodeIsCandidateMultiUnit() 117
- GeocodePostalCentroid() 117–118
- GeocodePostalCentroidWithSerial() 118–119
- GetCandidateAt() 119
- GetCandidateCensusBlockIDAt() 119
- GetCandidateCityAt() 120
- GetCandidateFirmAt() 120
- GetCandidateLatitudeAt() 120
- GetCandidateLongitudeAt() 121
- GetCandidatePlus4At() 121
- GetCandidatePrecisionAt() 122
- GetCandidateResultCodeAt() 122
- GetCandidateStateAt() 122
- GetCandidateStreetAt() 123
- GetCandidateZIPAt() 123
- GetFullName() 124
- GetName() 124
- GetVersion() 124
- RefreshDialog() 124
- SelectCandidateAt() 125

OLE automation objects
- CAddressList 125

operating systems 15

Oracle Geocoding Cartridge 31

P

performance
- increasing 17

positional accuracy of points
- result code information 46

postal geocoding 41

preferences
- generic and U.S. 43
- geocoding 39
- usage examples 44
- user dictionary 76–78

R

RefreshDialog() 124

re-installing MapMarker 24–25

remote database drivers 25

result codes
- definition 38
- interpreting 45–46
- retrieving 54

RPC Server API 29–30

running existing applications 32–34

S

sample application 65–66

SelectCandidateAt() 125

sending geocode requests 45

servlet deployment 69

setting
- geocoding constraints 50–52
- input address 49

Solaris applications
- JNI adapter 33
- migration path 30

starting MapMarker 26

street-level geocoding 41

Sun One 7.0 16, 69

system recommendations 16

system requirements 15–16

T

Tomcat Servlet Container 69

transition to MapMarker Plus 11.0 14, 31

U

UNIX applications
- JNI adapter 33–34
- migration path 30

user dictionaries
- configuring 76
- creating 72–75
- definition 39
- preferences 76–78

V

Visual Basic 6.0 API 29
W

war files 69
Web environments 16, 69
Web geocoding client 54–58
WebLogic 8.1 16, 69
WebSphere 5.1 and 6.0 16, 69
Windows console application
  running MapMarker server 87
Windows geocoding applications
  JNI adapter 32
  migration path 29–30
Windows service
  running MapMarker server 86–87