# Table of Contents

## Chapter 1: Introduction

- What Is Geocoding? .................................................. 10
- What Is MapMarker? ................................................... 10
- MapMarker Developer Product ...................................... 10
  - MapMarker Server and Developer Tools ......................... 10
  - MapMarker Desktop Application .................................. 11
- MapMarker Standard or MapMarker Plus .......................... 11
- New in This Release .................................................. 11
  - User Interface Enhancements .................................... 12
  - Address Dictionary Update ....................................... 12
  - MapInfo Professional Integration ............................... 12
- MapMarker Documentation Set ...................................... 12
  - MapMarker Developer Guide ..................................... 12
  - MapMarker Desktop User Guide .................................. 13
  - MapMarker Javadocs ............................................... 13
  - MapMarker Release Notes ........................................ 14
  - Publications on the Web ......................................... 14
  - Online Help ....................................................... 14
- MapMarker Streets .................................................. 14
- MapInfo Professional Integration .................................. 14
- Getting Technical Support .......................................... 15
  - Customer Service—Your Non-Technical Support ............... 18

## Chapter 2: Installing MapMarker .................................. 19

- Installation Overview ............................................... 20
  - Address Dictionary ................................................ 20
  - Upgrade Information ............................................. 20
- System Requirements ............................................... 20
  - Operating Systems ................................................ 20
  - Java Virtual Machine ............................................. 21
  - Supported Web Servers .......................................... 21
  - Minimum System Requirements ................................ 22
  - Recommended System Requirements ............................ 22
  - High Performance System Requirements ....................... 22
- Using MapMarker 12 WAR File in Tomcat 5.0 ..................... 22
- Configuring Tomcat 5.5 to Use Earlier JDK ....................... 22
- Address Dictionary ................................................ 23
- Administrator Rights ............................................... 23
- System Recommendations ......................................... 24
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storing Data on Your Hard Drive</td>
<td>24</td>
</tr>
<tr>
<td>Performance Tuning</td>
<td>24</td>
</tr>
<tr>
<td><strong>Windows Installation Procedures</strong></td>
<td>24</td>
</tr>
<tr>
<td>New Installations</td>
<td>24</td>
</tr>
<tr>
<td>Shared Installation</td>
<td>31</td>
</tr>
<tr>
<td>Upgrade Installations</td>
<td>33</td>
</tr>
<tr>
<td>Re-Installing MapMarker</td>
<td>33</td>
</tr>
<tr>
<td><strong>UNIX and Linux Installation Procedures</strong></td>
<td>34</td>
</tr>
<tr>
<td>GUI Installation</td>
<td>34</td>
</tr>
<tr>
<td>Silent Installation</td>
<td>34</td>
</tr>
<tr>
<td><strong>MDAC 2.8 and MapMarker ODBC Support</strong></td>
<td>35</td>
</tr>
<tr>
<td>Starting MapMarker</td>
<td>36</td>
</tr>
<tr>
<td>Starting the Desktop Product</td>
<td>36</td>
</tr>
<tr>
<td>Starting the Developer Product</td>
<td>36</td>
</tr>
<tr>
<td>Running Tomcat Server as a Windows Service.</td>
<td>36</td>
</tr>
<tr>
<td><strong>JNI Adapter Configuration File</strong></td>
<td>36</td>
</tr>
<tr>
<td>Startup Errors Related to JNI Adapter</td>
<td>37</td>
</tr>
<tr>
<td><strong>Chapter 3: Designing Geocoding Applications</strong></td>
<td>39</td>
</tr>
<tr>
<td>Choosing a Development Tool</td>
<td>40</td>
</tr>
<tr>
<td>MapInfo’s Geocoding Interoperability</td>
<td>40</td>
</tr>
<tr>
<td><strong>Geocoding Terms</strong></td>
<td>42</td>
</tr>
<tr>
<td>Defining the Geocoding Model</td>
<td>42</td>
</tr>
<tr>
<td>Geocoding Preferences</td>
<td>42</td>
</tr>
<tr>
<td>Use of Geocoded Data</td>
<td>43</td>
</tr>
<tr>
<td>Executing the Application</td>
<td>43</td>
</tr>
<tr>
<td><strong>Geocoding Application Design Overview</strong></td>
<td>43</td>
</tr>
<tr>
<td>Types of Geocoding</td>
<td>44</td>
</tr>
<tr>
<td>Building an Input Address</td>
<td>45</td>
</tr>
<tr>
<td>Using Preferences</td>
<td>46</td>
</tr>
<tr>
<td>Sending a Request</td>
<td>48</td>
</tr>
<tr>
<td>Verifying a Response</td>
<td>48</td>
</tr>
<tr>
<td>Using Geocode Result Information</td>
<td>48</td>
</tr>
<tr>
<td><strong>Chapter 4: Using C-Based Developer Tools</strong></td>
<td>51</td>
</tr>
<tr>
<td>MapMarker Geocoding API Development</td>
<td>52</td>
</tr>
<tr>
<td>Java Developers</td>
<td>52</td>
</tr>
<tr>
<td>Windows Developers</td>
<td>53</td>
</tr>
<tr>
<td>Solaris, HP-UX, and Linux Developers</td>
<td>53</td>
</tr>
<tr>
<td>Database Developers</td>
<td>53</td>
</tr>
<tr>
<td><strong>Client/Server Geocoding</strong></td>
<td>54</td>
</tr>
<tr>
<td>Installing MapMarker Server Manually</td>
<td>54</td>
</tr>
</tbody>
</table>
# Table of Contents

- Registering the OCX Manually .......................................................... 54
- MapMarker Server ................................................................................. 55
- Running MapMarker Server on Windows Systems ............................... 55
  - Windows Service .................................................................................. 55
  - Console Application .............................................................................. 56
  - Geocoding Request Timed Out .......................................................... 57
- Compiling and Running Existing Applications .................................... 57
  - Windows Applications ......................................................................... 57
  - Other Configuration File Settings ....................................................... 58

## Chapter 5: OLE Automation API. ......................................................... 59

- Creating a Custom Geocoding Control. ............................................... 60
- Geocoder Control Properties, Events, and Methods ............................ 61
- OLE Automation Methods ................................................................... 66
- ClearDialogText() ................................................................................ 68
- Connect() ............................................................................................ 69
- Disconnect() ....................................................................................... 70
- DoGeocode() ...................................................................................... 70
- DoSetProperties() ................................................................................ 70
- DpvGeocodeAddress() ......................................................................... 71
- DpvGeocodeAddressLastLine() ............................................................ 72
- DpvGeocodeAddressLastLineWithSerial() .......................................... 74
- DpvGeocodeAddressWithSerial() .......................................................... 75
- GeocodeAddress() .............................................................................. 77
- GeocodeAddressEx() ........................................................................... 80
- GeocodeAddressLastLine() .................................................................. 81
- GeocodeAddressLastLineEx() ............................................................... 83
- GeocodeAddressLastLineWithSerial() .................................................. 84
- GeocodeAddressWithSerial() ............................................................... 85
- GeocodeAddressWithSerialEx() ............................................................ 86
- GeocodeCheckDbAvailability() .............................................................. 88
- GeocodeFreeSet() ................................................................................ 88
- GeocodeGetCandidates() ..................................................................... 89
- GeocodeGetErrorText() ....................................................................... 90
- GeocodeGetHwyExitCandidate() ............................................................ 90
- GeocodeGetServerVersion() ................................................................. 92
- GeocodeGetStatesFound() ................................................................... 92
- GeocodeGetStatesLicensed() ................................................................. 93
- GeocodeHwyExit() ................................................................................ 93
- GeocodesCandidateMultiUnit() ............................................................. 95
- GeocodePostalCentroid() ..................................................................... 95
- GeocodePostalCentroidWithSerial() ..................................................... 96
Chapter 6: Using the MapMarker Java API.

Overview
Using the MapMarker Java USA API.
Using the MapMarker Generic Java API.

Developing Geocoding Applications in Java
Creating a Geocoding Application in Java.
Creating a Web Geocoding Client in Java.
Sample Code

Browsing Addresses

Geocoding to City Centroids
Geocoding to Highway Exits
Geocoding to Airports
Finding Airports by State
Finding Airports by Code

Address Point Interpolation

Chapter 7: MapMarker Sample Application
Requirements and Setup
Using the Sample Application
Testing the Server
<table>
<thead>
<tr>
<th>Chapter 8: Using the XML API</th>
<th>143</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapMarker XML Programming in .NET Framework</td>
<td>144</td>
</tr>
<tr>
<td>Interacting with a MapMarker Server</td>
<td>144</td>
</tr>
<tr>
<td>Using XSD Files to Create .NET Classes</td>
<td>144</td>
</tr>
<tr>
<td>Building a Sample Application</td>
<td>145</td>
</tr>
<tr>
<td>Setting Up the Project</td>
<td>145</td>
</tr>
<tr>
<td>Adding the Code</td>
<td>147</td>
</tr>
<tr>
<td>Sample Application Code</td>
<td>150</td>
</tr>
<tr>
<td>Example Class Modifications</td>
<td>152</td>
</tr>
<tr>
<td>Adding Objects to Collections in the Geocode Request</td>
<td>152</td>
</tr>
<tr>
<td>Request Web Headers</td>
<td>153</td>
</tr>
</tbody>
</table>

Chapter 9: Deploying Applications | 155 |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploying MapMarker as a Servlet</td>
<td>156</td>
</tr>
<tr>
<td>Using the Default Tomcat Servlet Container</td>
<td>156</td>
</tr>
<tr>
<td>Deploying MapMarker Server in Other Web Environments</td>
<td>156</td>
</tr>
<tr>
<td>Integration with MapMarker for Other Countries</td>
<td>156</td>
</tr>
</tbody>
</table>

Chapter 10: Creating and Using Custom Address Dictionaries | 159 |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating a Custom Dictionary</td>
<td>160</td>
</tr>
<tr>
<td>Required and Optional Fields</td>
<td>160</td>
</tr>
<tr>
<td>User Dictionary File Format</td>
<td>161</td>
</tr>
<tr>
<td>Data Dictionary Interface</td>
<td>162</td>
</tr>
<tr>
<td>GeoEngDDCreateUserDictionary()</td>
<td>162</td>
</tr>
<tr>
<td>Using Custom Dictionaries</td>
<td>164</td>
</tr>
<tr>
<td>Configuring User Dictionaries on the Server</td>
<td>164</td>
</tr>
<tr>
<td>User Dictionary Preferences</td>
<td>165</td>
</tr>
<tr>
<td>Client Preferences</td>
<td>167</td>
</tr>
<tr>
<td>J Server 3.1 to MapMarker Java 12.x Conversion</td>
<td>170</td>
</tr>
<tr>
<td>Text Conventions</td>
<td>170</td>
</tr>
<tr>
<td>Code Example</td>
<td>170</td>
</tr>
<tr>
<td>JVM Settings</td>
<td>176</td>
</tr>
<tr>
<td>Country and Coordinate System Settings</td>
<td>176</td>
</tr>
<tr>
<td>MapMarker Java API Errors</td>
<td>178</td>
</tr>
<tr>
<td>Engine Errors</td>
<td>178</td>
</tr>
<tr>
<td>Error Type</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Parser Errors</td>
<td>178</td>
</tr>
<tr>
<td>Data Access Exceptions</td>
<td>178</td>
</tr>
<tr>
<td>License Exceptions</td>
<td>179</td>
</tr>
<tr>
<td>General Geocoding Process Exceptions</td>
<td>179</td>
</tr>
<tr>
<td>Client Exceptions</td>
<td>180</td>
</tr>
<tr>
<td>Localization (Geocoder Construction Time) Exceptions</td>
<td>180</td>
</tr>
<tr>
<td>Server Exceptions</td>
<td>182</td>
</tr>
<tr>
<td><strong>Geocoding Engine Errors</strong></td>
<td>182</td>
</tr>
<tr>
<td><strong>Adapter Errors</strong></td>
<td>183</td>
</tr>
</tbody>
</table>
Welcome to MapMarker, MapInfo’s premier address matching product. MapMarker enables you to assign geographic coordinates to large tables of U.S.-based address records in a single session. In addition to being a powerful geocoder, MapMarker is also a suite of tools that allows you to standardize your U.S. addresses, add spatial information and create points for your records, develop standalone or client/server custom geocoding applications, and embed MapMarker functionality in existing applications.

In this chapter:

- What Is Geocoding? .................. 10
- What Is MapMarker? .................. 10
- MapMarker Standard or MapMarker Plus .................. 11
- New in This Release .................. 11
- MapMarker Documentation Set .................. 12
- MapMarker Streets .................. 14
- MapInfo Professional Integration .................. 14
- Getting Technical Support .................. 15
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 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.

**Note**  CASS and DPV functionality are available with MapMarker Plus.

**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.

**New in This Release**


The Release Notes also describe fixes to customer-reported issues and known issues that customers should be aware of.
User Interface Enhancements

The candidates list boxes in the Quick Find and Interactive Geocoding dialog boxes have been widened so that all the information about a candidate can be seen without having to scroll horizontally.

The data vintage and version information has been added to the About MapMarker dialog box (Help > About MapMarker).

Address Dictionary Update

The Address Dictionary has been updated with the latest data vintages. See the MapMarker release notes at http://reference.mapinfo.com/ for details.

MapInfo Professional Integration

MapInfo Professional users can run MapMarker or Envinsa 4.0 as a geocoding service from within MapInfo Professional 8.5. Connecting directly enables users to take advantage of MapMarker’s geocoding capabilities within MapInfo Professional’s powerful mapping and analysis environment.

To use this feature, you must have MapInfo Professional 8.5 and be running MapMarker 11.0 or later (in server mode) or Envinsa 4.0. For complete information on connecting to these services, see your MapInfo Professional 8.5 documentation.

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 PDF format
• Online Help for the MapMarker desktop product
• MapMarker Developer Guide in PDF format
• Release Notes that provide updated information on new features, behavioral changes in the software, fixes for customer-reported issues, and known issues

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.
Chapter 1: Introduction

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.

The Desktop User Guide 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.

**Note**  
CASS and DPV are available with MapMarker Plus only.

This guide also provides appendixes that provide reference information for the MapMarker desktop product.

- 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 USA Java API” in the box labeled MapMarker Documentation.
MapMarker Release Notes

The MapMarker Release Notes contain a brief summary of:

- new features
- behavioral changes
- bug fixes
- known issues

Publications on the Web

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

Documentation on the web site may be updated if corrections are necessary or if new information becomes available.

Online Help

In addition to the User Guide, MapMarker Standard and MapMarker Plus include Online Help for the desktop product. 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.

MapInfo Professional Integration

MapInfo Professional users can now run MapMarker Java server or Envinsa 4.0 as a geocoding service from within MapInfo Professional v. 8.5 or later.

MapInfo Professional users can take advantage of both single address as well as table geocoding capabilities, using MapMarker’s greater parsing, matching, and positional accuracy methodologies. MapInfo Professional users can also set multiple match conditions and user preferences.
Other advantages to using a geocoding service with MapInfo Professional include:

- **Multiple Data Formats.** Since MapInfo Professional can import or open data in many different formats, you can geocode almost any kind of geographically enabled file. You can take advantage of MapMarker or Envinsa’s advanced geocoding options using Shapefiles, Excel files, ASCII, Access tables, Oracle and SQL Server tables.
- **Data filtering.** You can use the selection and subselection capabilities of MapInfo Professional to create input for the service using any MapInfo Professional "table," including queries created using SQL Select or tools. For example, if you want to geocode your data based on more than one column, you can specify an input address based on a MapBasic expression that could skip irrelevant characters.
- **Geocode a Little or a Lot.** You can geocode individual records or in batch mode.
- **Choose your own Symbols.** You have the full range of MapInfo Professional symbols to choose from when plotting your points, or you can create your own custom symbols for the points you are geocoding.
- **Undo works.** Because all of the geocoded results are transacted, you can use MapInfo Professional’s revert capabilities to undo the transactions. You can save the geocoded results in the source table or into a completely new table.

**Note**  Expressions cannot be used for output.

---

**Getting Technical Support**

MapInfo Corporation offers a free support period on all new software purchases and upgrades, so you can be productive from the start. Once the free period ends, MapInfo Corporation offers a broad selection of extended support services for individual, business, and corporate users.

Technical Support is here to help you, and your call is important. This section lists the information you need to provide when you call your local support center. It also explains some of the technical support procedures so that you will know what to expect about the handling and resolution of your particular issue.

MapInfo Corporation provides full technical support for MapMarker for versions 10.0 and later. Please remember to include your serial number, partner number or contract number when contacting Technical Support.

Contact the technical support personnel for your area:

**The Americas**

Phone: 518.285.7283
Fax: 518.285.6080
E-mail: techsupport@mapinfo.com

Hours: Monday - Friday from 8:00am - 7:00pm EST, excluding MapInfo Holidays. Closed between 10:30am - 11:30 am on Mondays for training.
Getting Technical Support

Asia-Pacific Headquarters
Phone: 61.7.3844.7744
Fax: 61.7.3844.2400
E-mail: ozsupport@mapinfo.com
Hours: Monday–Friday from 9:00am and 5:00pm (EST) Australian Eastern Standard Time, excluding MapInfo Holidays.

Europe/Middle East/Africa
Phone: 44.1753.848229
Fax: 44.1753.621140
E-mail: support-europe@mapinfo.com
Hours: Monday–Friday from 8am to 5pm GMT, excluding MapInfo Holidays.

Germany
Phone: +49 (0) 6142-203-400
Fax: +49 (0) 6142-203-444
E-mail: supportgermany@mapinfo.com
Hours: Monday–Friday from 9 am to 5 pm MEZ, excluding MapInfo Holidays.

To use Technical Support, you must register your product. This can be done very easily during installation. To receive more information on MapInfo’s technical support programs, contact a representative in your area or one of our technical support offices.

In the United States, call 1-800-FASTMAP for more information. To purchase MapInfo technical support or renew your current contract, please contact MapInfo Customer Service at 1-800-552-2511, and press 3 at the main menu, or send an e-mail at custserv@mapinfo.com.

Extended support options are available at each of our technical support centers in the United States, United Kingdom, and Australia.

Before You Call

Please have the following information ready when contacting us for assistance on MapMarker.

1. Serial Number. You must have a registered serial number to receive Technical Support.

2. Your name and organization. The person calling must be the contact person listed on the support agreement.

3. Version of the product you are calling about.

4. The operating system name and version.

5. A brief explanation of the problem. Some details that can be helpful in this context are:
   • Error messages
   • Context in which the problem occurs
   • Consistency – is the problem reoccurring or occurring erratically?
The Support Tracking System

The Support Tracking System is used internally by the Technical Support department to manage and track customer issues. The system also provides the ability to track calls with accountability. This system helps Tech Support respond to all customer issues effectively, efficiently, and fairly.

Expected Response Time

Most issues can be resolved during the customer’s initial call. If this is not possible, a response will be issued before the end of the business day. A Technical Support representative will provide a status each business day until the issue is resolved.

Support requests submitted by e-mail are handled using the same guidelines as telephone support requests; however, there is an unavoidable delay of up to several hours for message transmission and recognition.

Exchanging Information

Occasionally a Technical Support representative will ask you to provide sample data in order to duplicate your scenario. In the case of our developer tools (such as MapXtreme), a small subset of sample code may be requested to help duplicate the issue.

The preferred method of exchanging information is either via e-mail or our FTP site. Use the following e-mail addresses:

- United States – techsupport@mapinfo.com
- Europe – support-europe@mapinfo.com
- Australia – ozsupport@mapinfo.com

Accessing the MapInfo FTP site

For information regarding our FTP site, please contact Technical Support. If information cannot be provided electronically, we also accept information in the following media formats:

- CD
- DVD

Software Defects

If the issue is deemed to be a bug in the software, the representative will log the issue in MapInfo Corporation’s bug base and provide you with an incident number that can be used to track the bug. Future upgrades and patches have fixes for many of the bugs logged against the current version.
Customer Service—Your Non-Technical Support

To receive resolve questions about the accounting of your MapInfo product, contact the Customer Service department. Keep in mind that this is not a technical support resource.

- Hours: Monday through Friday, 8AM to 7PM EST
- Telephone Number: (800) 552–2511, Option 3
- Queue/Voice mailbox: x6329
- Internet address: custserv@mapinfo.com
Installing MapMarker

This chapter contains instructions for installing MapMarker. Refer to this chapter for step-by-step instructions and other issues related to installation.

In this chapter:

- Installation Overview .................................................. 20
- System Requirements ..................................................... 20
- System Recommendations .............................................. 24
- Windows Installation Procedures ................................. 24
- UNIX and Linux Installation Procedures ...................... 34
- MDAC 2.8 and MapMarker ODBC Support ..................... 35
- Starting MapMarker ....................................................... 36
- JNI Adapter Configuration File ................................. 36
Installation Overview

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

Address Dictionary

The MapMarker Address Dictionary is installed during the installation of the MapMarker 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.

Upgrade Information

If you are updating your MapMarker 12.x installation, you can perform an upgrade install. For more information see Upgrade Installations on page 33.

System Requirements

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

Operating Systems

MapMarker runs on the following operating systems:

- Windows 2000 with SP4
- Windows XP Professional with SP2
• Windows 2003 Enterprise Edition

Java API
• Windows 2000 Advanced Server with SP4
• Windows XP Professional with SP2
• Windows 2003 Enterprise Edition
• Solaris 2.9, 2.10
• HP-UX 11.0
• Linux kernel version 2.6

C API
• Windows 2000 with SP 4
• Windows XP Professional with SP2
• Windows 2003 Enterprise Edition
• Solaris 2.9, 2.10
• HP-UX 11.0

Java Virtual Machine
MapMarker requires a Java Virtual Machine (JVM) to run. The minimum supported JVM is:
• JVM v. 1.4.2 or later
MapMarker ships with JVM v. 1.5_09.

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

JVM Requirements and Performance
MapMarker produces an error during the installation process when you select a JVM v. 1.4.1 or earlier. The following error message appears:

The virtual machine you have selected is not operational, or its version is not suitable for use with this product. Please select a java virtual machine that is version 1.4.2 or later.

Under high loads, MapMarker server may experience performance degradation with JVM v. 1.5.0_04 or earlier. For this reason, we recommend that you use JVM 1.5_05 or later when high server performance is required. JVM 1.5_09 ships with Mapmarker 12.

Supported Web Servers
The following Web servers are supported:
• Tomcat 5.5 (installed by MapMarker 12)
• Sun Application Server 9.0
• BEA WebLogic 9.0
• WebSphere 6.0
System Requirements

Minimum System Requirements

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

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

Recommended System Requirements

The recommended system requirements on a full are:

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

High Performance System 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 Address Dictionary requires approximately 2.63 GB (for nationwide data) of disk storage space. The DPV data (for MapMarker Plus) requires approximately .5 GB of disk storage space.

Using MapMarker 12 WAR File in Tomcat 5.0

File I/O errors can occur when you are using a MapMarker 12 WAR file in Tomcat if the Tomcat Java heap size is not allocated enough RAM.

MapMarker 12 requires a minimum of 256 RAM. This amount of RAM is automatically set when Tomcat 5.5 is installed during the MapMarker 12 installation. If you are deploying to a WAR file into your own Tomcat, then you must manually adjust the heap size to a minimum of 256 RAM.

Configuring Tomcat 5.5 to Use Earlier JDK

The MapMarker 12 Developer product installs Apache Tomcat 5.5, and that version of Tomcat is configured to run on J2SE 5.8 (JDK 1.5_08) or later. If you want to use an older JVM (for example, 1.4.2), Tomcat 5.5 won’t start because that server is configured with a newer version of the JVM. You will receive an error message when you try to start the server.

If you want to run Tomcat 5.5 on an earlier JVM, you must download and install a compatibility package from the Apache Tomcat binary download page.
You can get the Apache Tomcat compatibility package from one of the following links:

Zip: [http://mirrors.ccs.neu.edu/Apache/dist/tomcat/tomcat-5/v5.5.17/bin/apache-tomcat-5.5.17-compat.zip](http://mirrors.ccs.neu.edu/Apache/dist/tomcat/tomcat-5/v5.5.17/bin/apache-tomcat-5.5.17-compat.zip)

tar.gz: [http://mirrors.ccs.neu.edu/Apache/dist/tomcat/tomcat-5/v5.5.17/bin/apache-tomcat-5.5.17-admin.tar.gz](http://mirrors.ccs.neu.edu/Apache/dist/tomcat/tomcat-5/v5.5.17/bin/apache-tomcat-5.5.17-admin.tar.gz)

The package consists of three jar files that must be copied into the MapMarker tomcat directory. To copy the files:

1. Copy the following file to the tomcat directory:
   
   SDK/MapMarker-4.0.0/tomcat/bin/jmx.jar

2. Create a new folder named endorsed for the next two jar files.

3. Copy the following files to the endorsed folder:
   
   SDK/MapMarker-4.0.0/tomcat/common/endorsed/xercesImpl.jar
   
   SDK/MapMarker-4.0.0/tomcat/common/endorsed/xml-apis.jar

The compatibility package will add the following APIs to the common/lib or shared/lib folder. These APIs are needed when you run on J2SE 1.4:

- jmx.jar (Java Management Extensions API 1.2 or later)
- xercesImpl.jar (Xerces XML Parser, version 2.6.2 or later)

To override the XML parser implementation or interfaces, use the endorsed mechanism of the JVM. The default configuration defines JARs located in "common/endorsed" as endorsed.

**Address Dictionary**

The Address Dictionary must be stored on a local hard drive or a network drive. The data resides in the data directory under the MapMarker program directory (default is c:\Program Files\MapInfo\MapMarker_USA\data).

**Administrator Rights**

For installation on Windows operating systems, you must have administrator rights on the target machine to install MapMarker. If you attempt to install on a machine where you do not have administrator rights, the following error message appears:
Note the following recommendations in order to maximize the performance of your MapMarker 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 Procedures**

This section explains the MapMarker installation procedures for Windows operating system users.

**New Installations**

If you do not have a previous version of MapMarker installed, do the following:

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 dialog box appears.

2. Click **Install Product** in the list of options.

3. In the **Install Product** dialog box, click **Install MapMarker**.

4. In the Introduction dialog, click **Next** to display the License Agreement dialog box.

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 appears.
6. 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 appears.

7. In the Enter Unlocking Code dialog box, 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. 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 multi-state pack, you will be able to install those states after entering your unlocking code.

    Click Next to display the Choose Install Set dialog box.
8. In the Choose Install Set dialog box, the default Typical Install Set shows all the software and all the data you purchased. 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, click **Custom** in the Install Set drop-down list. Clear the check boxes of the components that you do not want to install.

Click **Next** to display the Installation Type dialog box.

9. In the Installation Type dialog box, select either **Single** or **Shared**. Select Single if your installation of MapMarker is a local installation of the software and data.

Select Shared if this installation of MapMarker will serve as a host and provide a group of users on the same network a common location for the software. In a shared installation, only the application files, client setup program, and data will be installed.

Those who wish to access the shared installation will need to perform a client installation. See **Shared Installation on page 31**.

10. If you selected the Shared button, a dialog will display, asking you if you would like to set up the MapMarker desktop application on this machine.

If you plan to use the MapMarker application on this machine, click **Yes**. If you are installing the software only for other users to access, click **No**. The Choose Software Install Folder dialog appears.
11. In the Choose Software Install Folder dialog box, select the location of the MapMarker software. You can use the default location:

C:\Program Files\MapInfo\MapMarker\MapMarker_v12

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 Folder dialog box.

Click **Next** in the Choose Software Install Folder dialog to display the Choose a Data Folder dialog box.
12. In the Choose a Data Folder dialog box, select the location of the MapMarker data. You can use the default location:

C:\Program Files\MapInfo\MapMarker\MapMarker_v12\Data

or click Choose to display the Browse for Folder dialog box, 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. Click Next to continue.

13. If you indicated that this installation would be used by others on a network, the Shared Install Information dialog displays. Continue to step step 14. If you indicated that this installation would be for a single user, skip to step 17.
14. At the Shared Install Information dialog, confirm or correct the existing path to the Address Dictionary. Universal Naming Convention (UNC) paths are recommended, as suggested in the dialog box. This ensures that users accessing MapMarker Plus from the shared machine will also be able to access the data.

15. Click **Next** to display the Choose Java Virtual Machine dialog box.

16. In the Choose Java Virtual machine dialog box, 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. 1.4.2 or later. For performance reasons, we recommend using at least JVM 1.5_05. See **JVM Requirements and Performance on page 21**. ships with JVM v. 1.5_09.

   **Note** If you select the Choose a Java VM already installed on this system button and there is no JVM installed on your computer, you will receive an error message when you click **Next**.
17. Click **Next** to continue. The Sample Web Environment parameters dialog box appears. 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.

![Sample Web Environment parameters dialog box]

18. Click **Next** to continue. The Choose Shortcut Folder dialog box appears. 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.

![Choose Shortcut Folder dialog box]
19. Click **Next** to continue. The Pre-Installation Summary dialog appears.

20. Review your install selections. To edit any of the selections, click the **Previous** button to go back to the earlier dialogs.

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

   An installation of MapMarker and the entire Address Dictionary may be time consuming. The amount of time required to install depends on how much data you are installing and machine processing power. Do not cancel the install process if there is hard drive or DVD drive activity. To verify that the installation is progressing, check the directory where the data is targeted to reside. You should see new files being added as the process continues.

22. When the installation is finished, the Install Complete dialog box appears. Click **Done** to exit the installer.

**Shared Installation**

The MapMarker data can be shared among a group of users. The system administrator performs a shared installation of MapMarker (which includes the Address Dictionary) in a common network location. Once the shared installation is complete, the system administrator must share the program folder and data folder across the network. Client users can browse to the program location and run the ClientSetup.exe to install a MapMarker client.

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

**Client Installation**

To set up a client workstation to access MapMarker from a shared installation on a network server, follow the steps in this section.
1. Browse to the location of the shared installation of MapMarker. (You may need to verify the location with your system administrator.)

2. From the shared machine, run the setup program (ClientSetup.exe) in the MapMarker program directory. The Welcome dialog appears.

3. At the Welcome dialog, click **Next** to display the Destination Folder dialog
   
   You can use the default location:
   
   C:\Program Files\MapInfo\MapMarker\
   
   or click the **Change** button to display the Change Current Destination Folder dialog and navigate to another location.

   ![Destination Folder Dialog](image)

   Click **OK** in the Change Current Destination Folder dialog when you have finished specifying the location.

4. Click **Next** display the Ready to Install the Program dialog box.

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

6. Click **Finish**.

   All the components necessary to run MapMarker as a client are now installed. This includes the data, as well as MDAC 2.8 and the Microsoft Jet engine with their desktop ODBC drivers.

   The installer creates a MapMarker program icon on the client machine, places appropriate system files in the Windows system directory, and updates the registry.
Chapter 2: Installing MapMarker

Upgrade Installations

If you are updating your installation from an earlier version of MapMarker Plus 12.x, and you have not purchased different data, you can perform an upgrade installation. Do the following:

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:\LAUNCH.EXE in the Open command box, where D is the drive letter of your DVD-ROM. Click OK. The MapMarker Installation Options appears.

2. Click Install Product in the list of options.

3. A message dialog box appears, asking you to either upgrade your installation or exit the installer.

4. Click Upgrade to proceed with the installation. The installer begins copying files. This may take a few minutes.

5. When the Upgrade Complete dialog box appears, click Done.

Re-Installing MapMarker

If you purchase components and data subsequent to your MapMarker purchase, such as licenses for additional states or DPV, 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 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 box appears. Your serial number is already filled in.
UNIX and Linux Installation Procedures

4. Click Next to continue. The Enter Unlocking Code dialog box appears. The unlocking 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 installing DPV as an add-on.
• You are installing additional data.

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, see Windows Installation Procedures on page 24.

UNIX and Linux Installation Procedures

This section explains the MapMarker installation procedures for UNIX and Linux operating system users.

GUI Installation

The installation process for UNIX closely resembles the installation procedure for Windows. To install the MapMarker Plus USA server on a Solaris, Linux, or HP-UX operating system:

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

1. A shell script starts the installation regardless of your operating system. Run it from the root of the CD-ROM using the command:

   sh startinstall.sh

2. To bypass the startinstall.sh script, go to the /InstData folder, and running the executable appropriate for your operating system. For example, on a Linux operating system:

   sh ./setupLinux.bin

3. At this point, the installation mirrors the Windows installation procedure. Follow the prompts, and refer to the Windows Installation Procedures on page 24 for specific information.

Silent Installation

MapMarker comes with a properties file located in the /instdata folder on the DVD-ROM that enables you to install MapMarker in silent mode:

   silentinstall.properties
This ASCII text file defines default values for the installer to use when installing without a GUI. You must edit the file to suit your particular needs. For example, the serial number and unlock code values are blank in the file. They must be populated with valid values in order for the install to execute properly. The file is configured to install the engine, web application, samples, additional tools, and licensed data.

**Note** The installation of the MapMarker Plus desktop application has been disabled for the silent install.

When you have completed the modifications to the file, you then pass in the properties file when you run the silent install.

To edit the properties file:

1. Copy the properties template file to a folder where you can edit it, and open the file in a text editor.
2. In the properties file, specify information for the install properties and set the software and data features that you want to install. The file contains instructions for changing each of the values. You must specify the following:
   - Serial number
   - Unlock code
   - Install location of the application files
   - Install location of the data
   - Web Server name
   - Startup port
   - Shutdown port
   - Location of symbolic links

To launch the silent install, use the following command line arguments:

```
sh /<path-to-install-media>/startinstall.sh -f <path-to-properties-file>/<property-file-name>
```

**MDAC 2.8 and MapMarker ODBC Support**

Microsoft Data Access Components 2.8 is required to use MapMarker with a SQL database. This component is installed automatically when you install MapMarker Plus 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 2.8 installer together install a number of ODBC drivers for use with MapMarker.

Among them are the following supported drivers:

- SQL Server 2000
- Microsoft Access 2000(*.mdb)

MapMarker also supports the following Oracle driver:

- Oracle 10G Release 1
You must install the Oracle driver separately. To use the driver, you must have the Oracle 10G client installed.

Starting MapMarker

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

Starting the Desktop Product

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

For information on possible startup errors associated with the JNI adapter, see Startup Errors Related to JNI Adapter on page 37.

Starting the Developer Product

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\tomcat\bin folder.

Running Tomcat Server as a Windows Service

To run the Tomcat Server as a Windows Service, do the following:

1. Open a DOS window and navigate to the MapMarker Tomcat bin folder. For example:
   
   C:\Program Files\MapInfo\MapMarker\MapMarker_USA\SDK\tomcat\bin

2. At the DOS command prompt, enter the following:
   
   type> service.bat install MM


4. Use the Java Sample application to test the service.

For more detailed information, see the Apache Web site:

http://jakarta.apache.org/tomcat/tomcat-5.5-doc/windows-service-howto.html

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/geoji.cfg

The environment variable can also be set by running the mme.profile file found in the SDK\additional_tools directory.

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.

**Startup Errors Related to JNI Adapter**

When you are using the MapMarker desktop product or the JNI adapter, you may encounter some startup errors. The following are explanations of some common errors and how to fix them.

**Error Code 1795**

JVM issue caused by selecting an unsupported JVM during the installation process.

MapMarker requires a JVM version 1.4.2 or later. By default, MapMarker installs a version 1.5_09 JVM.

To correct the error, perform a full uninstall of MapMarker and then reinstall it using the default JVM.

**Error Code 1871**

JVM issue caused by an improper JVM path being set in the geoji.cfg file (located in the MapMarker home folder).

To correct the error, fix the JVM.DLL path referenced in the geoji.cfg file, or uninstall MapMarker and then reinstall it using the default JVM.

```#
# geoji.cfg
#
# This file defines the jni settings and the
# MapMarker Java defaults.
#
# Path to the jvm.dll
+jvmPath=C:\Program
Files\MapInfo\MapMarker\MapMarker11\jre_USA\bin\server\jvm.dll
```

For more specific information about the configuration file settings, see [JNI Adapter Configuration File Settings in Appendix B on page 175](#).
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 ........................................40
- Geocoding Terms .........................................................42
- Defining the Geocoding Model .....................................42
- Geocoding Application Design Overview .......................43
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.

![Diagram of MapInfo's Geocoding Interoperability]

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.

**MapMarker 12.x**
- Desktop and Developer products. The Desktop includes the MapMarker 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.

**MapXtreme**
Developers using MapXtreme:
- Create Windows desktop or IIS web applications
- Use .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
- Use .Net, Java, XML or web service programming languages
- May require several location based capabilities (for example., reverse geocoding, which accepts an x, y position and returns address, intersection, postcode, city, or other location attribute information).

<table>
<thead>
<tr>
<th>Product</th>
<th>Web/ Desktop*</th>
<th>OS</th>
<th>Programming Language</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapMarker 12.x</td>
<td>Both</td>
<td>Windows, Solaris, HP-UNIX, Linux</td>
<td>Java</td>
<td>Geocoding</td>
</tr>
<tr>
<td>MapXtreme</td>
<td>Both (IIS)</td>
<td>Windows</td>
<td>.NET</td>
<td>Geocoding, Mapping, Routing</td>
</tr>
<tr>
<td>Envinsa</td>
<td>Both (App Server)</td>
<td>Windows, Solaris, Linux</td>
<td>Java, XML, .NET, and Web Services</td>
<td>Geocoding, Mapping, Routing, and more</td>
</tr>
</tbody>
</table>
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 46.

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
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>
</tbody>
</table>

*Country (required for generic)

*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.

<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>
</tbody>
</table>
| Street Intersection     | Street and ZIP Code, or street, city, and state  
Street 1 and street 2 names must be separated by double ampersands “&&” (e.g., “Global View && Jordan Road”, “42nd && Park”) | |
| Postal Centroid         | ZIP Code and/or ZIP + 4 | Country required in generic address structure |
| Geo                     | City and State        | Available through Java API |

**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>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>
</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.

<table>
<thead>
<tr>
<th>Locate</th>
<th>Address</th>
<th>PostCode</th>
<th>Town</th>
<th>Region</th>
<th>ResultCode</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps</td>
<td>350 OCEAN DR</td>
<td>33149</td>
<td>KEY BISCAYNE</td>
<td>FL</td>
<td>S5HFPNTSCA</td>
<td>true</td>
</tr>
<tr>
<td>Maps</td>
<td>350 OCEAN BLVD</td>
<td>33160</td>
<td>GOLDEN BEACH</td>
<td>FL</td>
<td>S2HFN S-A</td>
<td>false</td>
</tr>
<tr>
<td>Maps</td>
<td>1900 S OCEAN DR</td>
<td>33160</td>
<td>NORTH MIAMI BEACH</td>
<td>FL</td>
<td>S5-NTS-A</td>
<td>false</td>
</tr>
<tr>
<td>Maps</td>
<td>9200 OCEAN CURVE DR</td>
<td>33189</td>
<td>MIAMI</td>
<td>FL</td>
<td>S5-P1S-A</td>
<td>false</td>
</tr>
<tr>
<td>Maps</td>
<td>9200 OCEAN CURVY</td>
<td>33189</td>
<td>MIAMI</td>
<td>FL</td>
<td>S5-PNS-A</td>
<td>false</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Locate</th>
<th>Address</th>
<th>PostCode</th>
<th>Town</th>
<th>Region</th>
<th>ResultCode</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps</td>
<td>350 OCEAN DR</td>
<td>33149</td>
<td>KEY BISCAYNE</td>
<td>FL</td>
<td>S5HFPNTSCA</td>
<td>true</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Locate</th>
<th>Address</th>
<th>PostCode</th>
<th>Town</th>
<th>Region</th>
<th>ResultCode</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps</td>
<td>350 OCEAN DR</td>
<td>33149</td>
<td>KEY BISCAYNE</td>
<td>FL</td>
<td>S5HFPNTSCA</td>
<td>true</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Locality</th>
<th>Address</th>
<th>PostalCode</th>
<th>Town</th>
<th>Region</th>
<th>ResultCode</th>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>33149</td>
<td>----</td>
<td></td>
<td></td>
<td>21</td>
<td>true</td>
</tr>
</tbody>
</table>

**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.

**Note** For detailed information on MapMarker result codes, please see the Result Codes chapter in the MapMarker 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 House or address number match (e.g. 115)
- The fourth position describes street Prefix directional match (e.g. North)
- The fifth position describes street Name or main address match (e.g. School)
- The sixth position describes street Type match (e.g. Avenue)
- The seventh position describes street Suffix Directional match (e.g. NE)
- The eighth position describes City match (e.g. Miami)
- The ninth position describes Zip 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 C-Based Developer Tools

This chapter describes the support available for developers who use C-based developer tools and explains how to run the MapMarker Server as well as existing C-based geocoding applications in MapMarker USA.

In this chapter:

- MapMarker Geocoding API Development ......................... 52
- Client/Server Geocoding ........................................... 54
- MapMarker Server ..................................................... 55
- Running MapMarker Server on Windows Systems .......... 55
- Compiling and Running Existing Applications ............... 57
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 for Java developers.

Because both MapMarker USA and MapMarker Canada are now built on Java engines and utilize a full Java API, MapInfo made a decision to retire older Java tools. We strongly recommend that new applications be written using the Java API. The Java API offers a full range of geocoding functionality and is supported on Windows, UNIX, and Linux platforms.

For client/server geocoding, MapMarker USA supports the MapMarker Server and MapMarker Geocoder Control. Developers can also create a custom OCX for use with the MapMarker Server using the OLE Automation API. For more information on these tools, please see Client/Server Geocoding on page 54 and OLE Automation API in Chapter 5 on page 59.

Existing C applications can be run with the MapMarker Java API using the JNI adapter. For information on compiling and running existing applications, see Compiling and Running Existing Applications. on page 57.

The remainder of this section summarizes the tools and APIs that are available for each platform.

Java Developers

In the C-based version of MapMarker, Java developers who wanted to create geocoding solutions in Java needed to use the MapMarker J Server. Because both the MapMarker USA Java API and Envisna Java API provide full geocoding functionality and communicate directly with the Java engine, these APIs eliminate the need for the J Server API.

The following table summarizes the deprecation status of the J Server.

<table>
<thead>
<tr>
<th>Java API</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>J Server 2.x API</td>
<td>A servlet that uses JNI to access the C-based geocoding engine. J Server clients send requests via HTTP.</td>
<td>This API is no longer supported.</td>
</tr>
<tr>
<td>J Server 3.x API</td>
<td>A servlet that uses JNI to access the C-based geocoding engine. J Server clients send requests via HTTP.</td>
<td>Deprecated</td>
</tr>
</tbody>
</table>

Customers who move their applications from MapMarker J Server 3.x API to the MapMarker USA Java API will need to make minor code changes, but they will gain the advantages of utilizing a pure Java solution. For an example of converting a J Server application to MapMarker, see Converting MapMarker J Server Applications in Appendix A on page 169.
Windows Developers

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

<table>
<thead>
<tr>
<th>Windows API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE Automation API</td>
<td>The OLE Automation API is used to create customized OCX geocoding applications.</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>
</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>
</tr>
</tbody>
</table>

Developers can use their existing applications with the JNI adapter provided with MapMarker USA. The adapter enables existing C applications to be run with the MapMarker USA Java API.

Solaris, HP-UX, and Linux Developers

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

<table>
<thead>
<tr>
<th>API</th>
<th>Description</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>
</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>
</tr>
</tbody>
</table>

Developers can use their existing applications with the JNI adapter provided with MapMarker USA. The adapter enables existing C applications to be run with the MapMarker USA Java API.

Developers who want Linux support must use the MapMarker 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.
Client/Server Geocoding

<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>MapMarker Geocoding Cartridge for Oracle</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>An updated version of the Geocoding Cartridge ships with MapMarker. This release adds MapMarker USA functionality to the 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 USA 12.0 product ships. This update will add MapMarker USA functionality to the ESP SQL Server.</td>
</tr>
</tbody>
</table>

Client/Server Geocoding

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

MapMarker Server and the Geocoder Control are installed and registered on your system after you perform a full MapMarker installation.

Installing MapMarker Server Manually

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

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

  `mm_serve.exe` can be found in the `SDK\additional_tools` folder.

Registering the OCX Manually

If you need to register the OCX manually:

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

  `Regsvr32.exe` is located in the `System32` folder in your Windows directory. `mapmarkr.ocx` is found in `\SDK\additional_tools` folder on the target machine.
MapMarker Server

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

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

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

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:


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.


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\MapMarker\MapMarker_USA\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\USA\<version number>\SYSTEM.

Note In references to the MapMarker registry keys, <version number> indicates the major-release version of MapMarker, e.g., 12.0, even if you are running one of the point releases of that version, e.g., 12.1, 12.2.

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:

2. Click Stop.

To remove the MapMarker service from Windows Services:

• At the DOS prompt, type mmServe -delete.

Console Application

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

• At the DOS prompt type mmServe -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\MapMarker_USA\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\USA\<version number>\SYSTEM.

Note In references to the MapMarker registry keys, <version number> indicates the major-release version of MapMarker, e.g., 12.0, even if you are running one of the point releases of that version, e.g., 12.1, 12.2.
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**.

Customers who use version 4.0 of the Oracle Geocoding Cartridge can use the MapMarker Java API with MapMarker.

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

**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 C version of the GeoEngine API, and enables your existing applications to be run in MapMarker USA. 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\additional_tools\geoeng\include` folder where MapMarker is installed.

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

Link your application to `mm32v12.lib`. This file is located in the `SDK\additional_tools\` folder where MapMarker 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

The classpath in geojni.cfg is set at install time. The classpath in geojni.cfg should be 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_can.jar
- mmjclient_can.jar
- mmj_can.jar
- miscp.jar

**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 175](#).
This chapter explains the OLE Automation API. Use this to build your own geocoding control for your client application instead of using the program-ready MapMarker Geocoder Control (OCX)

In this chapter:

- Creating a Custom Geocoding Control ...........................................60
- Geocoder Control Properties, Events, and Methods ..................61
- OLE Automation Methods .................................................................66
- OLE Automation Objects .................................................................105
- Programming Usage Example .........................................................107
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**  To create a custom geocoding control in MapMarker 12.x, you must use the OLE Automation API. In MapMarker 12.x, calls are passed either directly through the JNI adapter to the MapMarker Java API, or to the RPC Server, and then through the JNI adapter to the MapMarker Java API.

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 geoengine mn32v_.dll. Default: False.</td>
</tr>
<tr>
<td>BCLOSECandidates</td>
<td>Boolean: If TRUE, shows close matches only; if FALSE shows all possible matches. Set this property on the Match Restrictions tab of the Property Pages dialog. Default: False.</td>
</tr>
<tr>
<td>ExactHouse</td>
<td>Boolean: If TRUE, exact match on house number is required Default: True.</td>
</tr>
<tr>
<td>ExactName</td>
<td>Boolean: If TRUE, exact match on street name is required Default: False.</td>
</tr>
<tr>
<td>ExactZIP</td>
<td>Boolean: If TRUE, exact match on ZIP Code is required Default: False.</td>
</tr>
<tr>
<td>ExactCity</td>
<td>Boolean: If TRUE, exact match on City is required. Default: False</td>
</tr>
<tr>
<td>ExpandSearch</td>
<td>Boolean: If TRUE, MapMarker expands the search area in which it looks for match candidates. Default: False.</td>
</tr>
<tr>
<td>ExpandSearchInState</td>
<td>Boolean: If TRUE, the expanded search area is limited to the state. Default: False.</td>
</tr>
</tbody>
</table>
### Input Properties

<table>
<thead>
<tr>
<th>Input Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<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>
</tbody>
</table>
### Chapter 5: OLE Automation API

#### Input Properties

<table>
<thead>
<tr>
<th>Input Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShowBorder</td>
<td>Boolean: If TRUE, the border around the control's interface is visible. Default: True.</td>
</tr>
<tr>
<td>CassMode</td>
<td>Boolean: If TRUE, CASSMode is set to on. Default: False</td>
</tr>
</tbody>
</table>

#### Output Properties

<table>
<thead>
<tr>
<th>Output Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitude</td>
<td>Double: The latitude value for a match candidate.</td>
</tr>
<tr>
<td>Longitude</td>
<td>Double: the longitude value for a match candidate.</td>
</tr>
</tbody>
</table>
| Precision            | Integer: A number that identifies match precision (street level, shape path, intersection, point ZIP, or ZIP centroid). No Centroid 0 | ZIP Point Centroid 1
ZIP + 2 Centroid 2 | ZIP + 4 Centroid 3
Shape Path Center 10 | Street Address 20
Street Intersection 30 | Point ZIP 40 |
| NumCandidates        | Integer: The number of match candidates for the record. |
| NumCloseCandidates   | Integer: The number of close match candidates for the record. |
| LastErrorCode        | Long: Returns the last error code generated. |
| CensusBlockID        | String: Census Block tabulation number. |
| ResultCode           | String: Geocoding result code. |
Geocoder Control Properties, Events, and Methods

## Output Properties

<table>
<thead>
<tr>
<th>Output Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 Service. If the match is from a TIGER record, the candidate may not have 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>
</tbody>
</table>
## Output Properties

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

### Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LatLongChanged()</td>
<td>This event is triggered when the Geocode button is pressed, the DoGeocode() method is called, or when a different candidate is selected in the Match Candidates list box. This event updates the latitude, longitude, and precision properties.</td>
</tr>
<tr>
<td>GeocodeEvent()</td>
<td>This event is triggered when the Geocode button is pressed and the whole geocoding process is completed.</td>
</tr>
</tbody>
</table>

**Note**: The DoGeocode() method does not trigger this event.
Chapter 5: OLE Automation API

- **GeocodeAddressLastLine()** ............................................. 81
  Enables you to enter an input address with an unparsed line containing the city, state, and/or ZIP Code.

- **GeocodeAddressLastLineEx()** ........................................... 83
  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()** ............................. 84
  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()** ......................................... 85
  Requires user to input a serial number for geocoding.

- **GeocodeAddressWithSerialEx()** ....................................... 86
  Requires user to input a serial number for geocoding, and enables street2 information to be passed in as a separate field.

- **GeocodeCheckDbAvailability()** ....................................... 88
  Checks to see which types of database(s) are available for geocoding.

- **GeocodeFreeSet()** .......................................................... 88
  Frees the server side information after geocoding.

- **GeocodeGetCandidates()** ............................................... 89
  Gets all the candidate information after geocoding.

- **GeocodeGetErrorText()** ................................................. 90
  Returns a short text description of the specified error.

- **GeocodeGetHwyExitCandidate()** ..................................... 90
  Gets the specified highway exit candidate after a call to GeocodeHwyExit_A().

- **GeocodeGetServerVersion()** .......................................... 92
  Returns the server version number.

- **GeocodeGetStatesFound()** ............................................. 92
  Returns a list of 2-digit state abbreviations representing states found in the Address Dictionary path. These are not necessarily licensed states.

- **GeocodeGetStatesLicensed()** ......................................... 93
  Returns a list of 2-digit state abbreviations representing licensed states.

- **GeocodeHwyExit()** ......................................................... 93
  Returns a count of the number of highway records based on input criteria of highway exit and state.

- **GeocodeIsCandidateMultiUnit()** .................................... 95
  Returns a True/False value, indicating whether the candidate address contains multiple units.

- **GeocodePostalCentroid()** .............................................. 95
  Geocodes a ZIP Code or ZIP + 4 centroid.

- **GeocodePostalCentroidWithSerial()** ................................ 96
  Requires a serial number for ZIP Code-level geocoding.

- **GetCandidateAt()** ........................................................... 97
  Returns the line of text in the Match Candidate list box.

- **GetCandidateCensusBlockIDAt()** .................................... 98
  Returns the Census Block ID as a string.

- **GetCandidateCityAt()** .................................................... 98
  Returns the city portion of the address as a string.
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 LastError Code 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 MapMarker Server is running. NULL or an empty string assumes MapMarker Server is running locally, and attempts to use local RPC.</td>
</tr>
</tbody>
</table>

Returns

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

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>NULL input parameter or invalid parameter values.</td>
</tr>
<tr>
<td>GEO_ENG_CORRUPTED_ERR</td>
<td>2</td>
<td>Memory overwrite detected.</td>
</tr>
<tr>
<td>GEO_ENG_DATABASE_ACCESS_ERR</td>
<td>10</td>
<td>Error accessing the Address Dictionary.</td>
</tr>
<tr>
<td>GEO_ENG_MISSING_DATABASE_ERR</td>
<td>1078</td>
<td>Address Dictionary file(s) not found.</td>
</tr>
<tr>
<td>GEO_ENG_ZIP_MASTER_FILE_NOT_FOUND_ERR</td>
<td>1228</td>
<td>Zipmastr.cdb or Zipmastr.jdx not found.</td>
</tr>
<tr>
<td>GEO_ENG_DBQ_MISSING_LICFILE</td>
<td>1079</td>
<td>License file not found.</td>
</tr>
<tr>
<td>GEO_ENG_INVALID_SERIAL_NUMBER</td>
<td>1085</td>
<td>Invalid serial number.</td>
</tr>
</tbody>
</table>
Disconnect()

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

**Syntax**

```vbnet
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**

```vbnet
DoGeocode()
```

---

DoSetProperties()

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

**Syntax**

```vbnet
DoSetProperties() As Boolean
```

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

---

### Errors

<table>
<thead>
<tr>
<th>Return</th>
<th>Return #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_ENGINE_PARSE_INIT_ERR</td>
<td>307</td>
<td>Parsing file (geo_usa.*) missing or bad.</td>
</tr>
<tr>
<td>GEO_ENGINE_MATCH_INIT_ERR</td>
<td>563</td>
<td>Matching file (geo_usa.*) missing or bad.</td>
</tr>
<tr>
<td>GEO_ENGINE_NADCON_MISSING_FILE</td>
<td>1334</td>
<td><em>.las/</em>.los files not found.</td>
</tr>
</tbody>
</table>
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. The CASSMode property must be set to True or the method will be unable to enable DPV mode.

Note  CASS and DPV functionality are available with MapMarker Plus.

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>
DpvGeocodeAddressLastLine()

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. The CASSMode property must be set to True or the method will be unable to enable DPV mode.

Note  CASS and DPV functionality are available with MapMarker Plus.

Parameter Description

<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>
## Syntax

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

## Parameters

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

Parameter | Description
--- | ---
status | [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.
numCandidates | [Output Integer] The total number of candidates found.
numCloseCandidates | [Output Integer] The number of close candidates.

Returns

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

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. The CASSMode property must be set to True or the method will be unable to enable DPV mode. This method should now be used instead of GeocodeAddressLastline, which used to have an optional serial number parameter.

Note  CASS and DPV functionality are available with MapMarker Plus.
### Syntax

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

### Parameters

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

### Returns

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

#### 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. The CASSMode property must be set to True or the method will be unable to enable DPV mode.

**Note** CASS and DPV functionality are available with MapMarker Plus.
DpvGeocodeAddressWithSerial()

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>
### GeocodeAddress()

#### Purpose

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>city</td>
<td>[Input String] City name to be matched.</td>
</tr>
<tr>
<td>state</td>
<td>[Input String] State to be matched.</td>
</tr>
<tr>
<td>zip</td>
<td>[Input String] Postal code to be matched.</td>
</tr>
</tbody>
</table>
Chapter 5: OLE Automation API

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

```
Private Sub cmdGeocode_Click()
    Dim retVal As Boolean
    Dim retCode
    retVal = objMM.GeocodeAddress(lngEngHandle, txtfirm, txtstreet, txtcity, txtstate, txtzip, status, numCandidates, numCloseCandidates)
```
GeocodeAddressEx()

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

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 77 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
GeocodeAddressLastLine()

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geocodeHandle</td>
<td>[Output Long] Used to reference information about the geocoding operation in other calls.</td>
</tr>
<tr>
<td>firm</td>
<td>[Input String] Firm name to be matched.</td>
</tr>
<tr>
<td>street</td>
<td>[Input String] Street name to be matched.</td>
</tr>
<tr>
<td>lastline</td>
<td>[Input String] Unparsed last line of address to be matched.</td>
</tr>
<tr>
<td>status</td>
<td>[Output Integer] One of the following:</td>
</tr>
<tr>
<td></td>
<td>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code (CASS mode only).</td>
</tr>
<tr>
<td></td>
<td>0 – 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>

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.

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

**Parameters**

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

**Purpose**

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

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

**Syntax**

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

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

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 107, 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
            For Each c in adr
                strAddr = strAddr & c & vbCrLf
            Next
            lstCandidates.AddItem strAddr
        End With
    Next
End Sub
GeocodeGetErrorText()

‘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 &
   trCensId)
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
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVersionNum</td>
<td>[Output Double] MapMarker version in integer format (i.e., 9)</td>
</tr>
</tbody>
</table>

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

**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>
<tbody>
<tr>
<td>pstateList</td>
<td>[Output String] A list of space-separated 2-digit state abbreviations</td>
</tr>
<tr>
<td></td>
<td>representing licensed states.</td>
</tr>
</tbody>
</table>

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

**GeocodeHwyExit()**

**Purpose**
This method uses input criteria of highway exit and state, and returns a count of the number of Hwy records that matched the input criteria. A subsequent call to GeocodeGetHwyExitCandidate() retrieves the matched candidates.

**Syntax**
GeocodeHwyExit(
    Hwyexit as String,
    State as String,
    count as Integer) As Boolean
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hwyext</td>
<td>[Input String] The highway and exit information in the form: &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.
Chapter 5: OLE Automation API

GeocodesCandidateMultiUnit()

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 GeocodesCandidateMultiUnit() function with the CAddressList, keep in mind that the index for GeocodesCandidateMultiUnit() 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>
<tr>
<td>Precision</td>
<td>[Output Integer] Returns one of the following: 40 (point ZIP); 30 (street-level match for an intersection address); 20 (street-level match for street address); 10 (Shape Path Centroid match); 3 (ZIP + 4 Centroid match); 2 (ZIP + 2 Centroid match); or 1 (ZIP Code Centroid match).</td>
</tr>
<tr>
<td>ResultCode</td>
<td>[Output String] Returns a value that is analogous to the GeoResult codes returned by MapMarker.</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
```
Chapter 5: OLE Automation API

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:</td>
</tr>
<tr>
<td></td>
<td>10 (Shape Path Centroid match);</td>
</tr>
<tr>
<td></td>
<td>20 (street-level match for street address);</td>
</tr>
<tr>
<td></td>
<td>30 (street-level match for an intersection address):</td>
</tr>
<tr>
<td></td>
<td>40 (point ZIP)</td>
</tr>
<tr>
<td></td>
<td>1 (ZIP Code Centroid match).</td>
</tr>
<tr>
<td></td>
<td>2 (ZIP + 2 Centroid match); or</td>
</tr>
<tr>
<td></td>
<td>3 (ZIP + 4 Centroid match);</td>
</tr>
<tr>
<td>ResultCode</td>
<td>[Output String] Returns a value that is analogous to the Georesult codes</td>
</tr>
<tr>
<td></td>
<td>returned by MapMarker.</td>
</tr>
<tr>
<td>SerialNumber</td>
<td>[Input String] Serial number required by server, created by an OEM license</td>
</tr>
<tr>
<td></td>
<td>program sold separately.</td>
</tr>
</tbody>
</table>

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

GetCandidateAt(Index As Integer) As String
GetCandidateCensusBlockIDAt()

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

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
# Chapter 5: OLE Automation API

## 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**

```vba
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**

```vba
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 firm of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur.
GetCandidateLongitudeAt()

**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
GetCandidateStateAt()

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.

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 state of the specified candidate. The LastErrorCode property can be used to determine the nature of any errors that occur.
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
    End If
Label8.Visible = False
Else
    txtlastline.Visible = False
Label21.Visible = False
txtCity.Visible = True
txtState.Visible = True
txtZip.Visible = True
txtZip4.Visible = True
Label15.Visible = True
Label16.Visible = True
Label17.Visible = True
Label18.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
        ElseIferrorCode = 1753 Then
            If txtServer = "" Then
                lblStatus = "Server : " & txtServer & " is not
running"
End If
Else
    lblStatus = "Error geocoding, error code: " &
        Str$(errorCode)
End If
Else
    lblLong = dblLong
    lblLat = dblLat
    Select Case intPrec
        Case 3
            lblPrec = "Zip+4 
        Case 2
            lblPrec = "Zip+2 
        Case 1
            lblPrec = "Zip centroid 
    End Select
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
End If
    retVal = objMM.GeocodeGetServerVersion(serverVersion)
    txtVersion = Str$(serverVersion)
    retVal = objMM.GeocodeGetStatesLicensed(txtBuf)
    txtLicensed = txtBuf
    retVal = objMM.GeocodeGetStatesFound(txtBuf)
    txtAvailable = txtBuf
End If
    cmdRefresh_Click
Exit Sub

Connect_Error:
If txtServer = "" Then
    MsgBox ("Error connecting to local server")
Else
    MsgBox ("Error connecting to server: " & txtServer)
End If
End Sub

Private Sub cmdDisconnect_Click()
Dim retVal As Boolean
If bConnected = True Then
    retVal = objMM.GeocodeFreeSet(lngEngHandle)
    retVal = objMM.Disconnect
    bConnected = False
    cmdGeocode.Enabled = False
    cmdCentroid.Enabled = False
    cmdGetCand.Enabled = False
    cmdConnect.Enabled = True
    cmdDisconnect.Enabled = False
    cmdFreeSet.Enabled = False
    DatabaseTypes = objMM.DatabaseTypes
End If
    cmdRefresh_Click
End Sub

Private Sub cmdFreeSet_Click()
Dim retVal As Boolean
    retVal = objMM.GeocodeFreeSet(lngEngHandle)
    'Candidates freed
    cmdGetCand.Enabled = False
    cmdFreeSet.Enabled = False
End Sub

Private Sub cmdGeocode_Click()
Dim retVal As Boolean
Dim retCode
Dim errorCode&
    ' User changed geocoding config parameters
    If bSettingModified = True Then
        retCode = MsgBox("Config parameters have been changed. Do you want to use new parameters?", 3, "Street Geocoding")
        If retCode = vbYes Then
            cmdSet_Click
        ElseIf retCode = vbCancel Then
            Exit Sub
        End If
    End If
    lblLong = ""
    lblLat = ""
    lblPrec = ""
    lblStatus = ""

lblCand = ""
lstCandidates.Clear
On Error GoTo GeocodeAddress_Error
If chklastline.Value = 1 Then
    retVal = objMM.GeocodeAddressLastline(lngEngHandle, 
        txtFirm, txtStreet, txtlastline, status, numCandidates, 
        numCloseCandidates)
Else
    retVal = objMM.GeocodeAddress(lngEngHandle, txtFirm, 
        txtStreet, txtCity, txtState, txtZip, status, numCandidates, 
        numCloseCandidates)
End If
On Error GoTo 0
If Not retVal Then
    errorCode = objMM.LastErrorCode
    If errorCode = 14 Then
        lblStatus = "Invalid address"
    'Sometimes .Connect returns true even when server is 
    'not running, but .GeocodeAddress will return 1753
    ElseIf errorCode = 1753 Then
        If txtServer = "" Then
            lblStatus = "Local server is not running"
        Else
            lblStatus = "Server : " & txtServer & " is 
            not running"
        End If
    Else
        lblStatus = "Error geocoding, error code: " 
        & Str$(errorCode)
    End If
Else
    DatabaseTypes = objMM.DatabaseTypes
    Select Case status
        Case 0
            lblStatus = "Single match"
        Case 1
            lblStatus = "Multiple match"
        Case 2
            lblStatus = "No close matches"
        Case 3
            lblStatus = "No candidates"
        Case 4
            lblStatus = "Single intersection match"
        Case 5
            lblStatus = "Multiple intersection match"
        Case 6
            lblStatus = "No close intersection matches"
        Case 7
            lblStatus = "No intersection candidates"
        Case 8
           lblStatus = "Possible intersections"
        Case Else
            lblStatus = Str$(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
            strAddr = .Street & "", " & .City & "", " & .State
            & " " & .Zip & "-" & .plus4
        Else
            strAddr = .Firm & "", " & .Street & "", " & .City
            & "", " & .State & " " & .Zip & "-" & .plus4
        End If
        strCoords = " (" & Format(.Latitude, "##0.0000") & ", ", " & Format(.Longitude, "##0.0000") & ")"
        Select Case .Precision
            Case 30
                strPrec = " Street-level (Xsect) "
            Case 20
                strPrec = " Street-level 
            Case 10
                strPrec = " Shape-path Cent. "
            Case 3
                strPrec = " Zip+4 
            Case 2
                strPrec = " Zip+2 
            Case 1
                strPrec = " Zip centroid 
        End Select
        strCensId = .CensusBlock
End With
Set adr = Nothing
lstCandidates.AddItem (strAddr & strCoords & strPrec & strCensId)

Next
Set adrList = Nothing
Exit Sub

GeocodeGetCandidates_Error:
MsgBox "Error getting candidates"
End Sub

Private Sub cmdRefresh_Click()
' Set UI from the properties of objMM
With objMM
    txtServer = .ServerName
    lblBinding = .StringBinding
    ExeName = .GetName
    If .ExactHouse = True Then
        chkMatchHouse.Value = 1
    Else
        chkMatchHouse.Value = 0
    End If
    If .ExactName = True Then
        chkMatchStreet.Value = 1
    Else
        chkMatchStreet.Value = 0
    End If
    If .ExactZip = True Then
        chkMatchCentroid.Value = 1
    Else
        chkMatchCentroid.Value = 0
    End If
    If .ExpandSearch = True Then
        chkExtendedSearch.Value = 1
    Else
        chkExtendedSearch.Value = 0
    End If
    If .ExpandSearchInState = True Then
        chkInState.Value = 1
    Else
        chkInState.Value = 0
    End If
    If .MatchIntersections = True Then
        chkMatchXsec.Value = 1
    Else
        chkMatchXsec.Value = 0
    End If
    txtDistance = .ExpandDistance
    txtOffsetLine = .LinearOffset
    txtOffsetPerp = .PerpendicularSetback
    bSettingModified = False
End With
End Sub

Private Sub cmdSet_Click()
    'Set the properties of objMM from values in the UI
    With objMM
        .ServerName = txtServer
        If chkMatchHouse.Value = 1 Then
            .ExactHouse = True
        Else
            .ExactHouse = False
        End If
        If chkMatchStreet.Value = 1 Then
            .ExactName = True
        Else
            .ExactName = False
        End If
        If chkMatchCentroid.Value = 1 Then
            .ExactZip = True
        Else
            .ExactZip = False
        End If
        If chkExtendedSearch.Value = 1 Then
            .ExpandSearch = True
        Else
            .ExpandSearch = False
        End If
        If chkInState.Value = 1 Then
            .ExpandSearchInState = True
        Else
            .ExpandSearchInState = False
        End If
        If chkMatchXsec.Value = 1 Then
            .MatchIntersections = True
        Else
            .MatchIntersections = False
        End If
        .ExpandDistance = txtDistance
        .LinearOffset = txtOffsetLine
        .PerpendicularSetback = txtOffsetPerp
        bSettingModified = False
    End With
End Sub

Private Sub Form_Load()
    Set objMM = CreateObject("MAPMARKR.MapMarkrCtrl.1")
    bConnected = False
    bSettingModified = False
    cmdGeocode.Enabled = False
    cmdCentroid.Enabled = False
    cmdGetCand.Enabled = False
    cmdFreeSet.Enabled = False
    cmdDisconnect.Enabled = False
Programming Usage Example

```vba
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.GetFileName
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
Using the MapMarker 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 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 ................................................................. 120
- Developing Geocoding Applications in Java .................. 120
- Browsing Addresses .................................................. 131
- Geocoding to City Centroids ...................................... 132
- Geocoding to Highway Exits ...................................... 133
- Geocoding to Airports ................................................. 134
- Address Point Interpolation ........................................ 136
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 Java USA API

The MapMarker 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 as a servlet along with other MapMarker countries see Integration with MapMarker for Other Countries on page 156.

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

Developing Geocoding Applications in Java

Use either the MapMarker USA Java API or the MapMarker Generic API to develop custom geocoding applications in Java. See Using the MapMarker Java USA API on page 120 and Using the MapMarker Generic Java API on page 120 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_USA\desktop 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 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 USA. It is located in the SDK\tomcat\webapps\ROOT\mapmarker_usa\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 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.setFallbackTo Postal(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 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>
</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_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>
<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>&quot;false&quot;</td>
</tr>
<tr>
<td>KEY_MUST_MATCH_AREA3</td>
<td>Only candidates matching a city or town are considered close.</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>KEY_MUST_MATCH_POSTAL</td>
<td>Only candidates matching postal code are considered close.</td>
<td>&quot;false&quot;</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>&quot;false&quot;</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>&quot;false&quot;</td>
</tr>
<tr>
<td>KEY_CORNEROFFSET</td>
<td>Defines the position of the geocoded point with respect to the corner.</td>
<td>&quot;7&quot;</td>
</tr>
<tr>
<td>KEY_CORNEROFFSETUNITS</td>
<td>The units used in KEY_CORNEROFFSET.</td>
<td>&quot;m&quot; (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>&quot;7&quot;</td>
</tr>
<tr>
<td>KEY_STREETOFFSETUNITS</td>
<td>The units used in KEY_STREETOFFSET.</td>
<td>&quot;m&quot; (meters)</td>
</tr>
</tbody>
</table>

### Table 6-1: USA_GeocodeConstraints

<table>
<thead>
<tr>
<th>Key</th>
<th>Base Equivalent</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_MUST_MATCH_ZIPCODE</td>
<td>POSTAL</td>
<td>Requires ZIP Code match for a close match.</td>
<td>&quot;false&quot;</td>
</tr>
<tr>
<td>KEY_MUST_MATCH_CITY</td>
<td>AREA3</td>
<td>Requires city/town match for a close match.</td>
<td>&quot;false&quot;</td>
</tr>
</tbody>
</table>
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 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.

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

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Base Equivalent</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_MUST_MATCH_STREETNAME</td>
<td>STREET</td>
<td>Requires street name match for a close match.</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_PREFER_USER_DICTIONARY</td>
<td></td>
<td>When using AD and UD together, prefer a UD close match over an AD close match</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_EXPANDED_SEARCH</td>
<td></td>
<td>Use an expanded search radius to find a match</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_EXPANDED_SEARCH_RADIUS</td>
<td></td>
<td>The distance to use for the expanded search radius</td>
<td>“25” mi</td>
</tr>
<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>

<table>
<thead>
<tr>
<th>Key</th>
<th>Base Equivalent</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_MUST_MATCH_STREETNAME</td>
<td>STREET</td>
<td>Requires street name match for a close match.</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_PREFER_USER_DICTIONARY</td>
<td></td>
<td>When using AD and UD together, prefer a UD close match over an AD close match</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_EXPANDED_SEARCH</td>
<td></td>
<td>Use an expanded search radius to find a match</td>
<td>“false”</td>
</tr>
<tr>
<td>KEY_EXPANDED_SEARCH_RADIUS</td>
<td></td>
<td>The distance to use for the expanded search radius</td>
<td>“25” mi</td>
</tr>
<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>
Chapter 6: Using the MapMarker Java API

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Datum</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longitude / Latitude</td>
<td>WGS 84</td>
<td>EPSG:4326</td>
</tr>
<tr>
<td>Longitude / Latitude</td>
<td>NAD 83</td>
<td>EPSG:4269</td>
</tr>
<tr>
<td>Longitude / Latitude</td>
<td>NAD 27 for Continental United States</td>
<td>EPSG:4267</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 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 126.

```java
/* 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 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 Desktop User Guide.

### Creating a Web Geocoding Client in Java

If you have deployed MapMarker 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 120, 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.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.GeocodeConstraints;
import com.mapinfo.mapmarker.IGeocodeConstraints;
import com.mapinfo.mapmarker.common.AddressImpl;
import com.mapinfo.mapmarker.common.Address;

Make sure that the jar files listed below are in your classpath. You can find the following files in the
SDK\engine\lib\client and SDK\engine\lib\common folder under the directory where you installed the
product.

- miutil.jar
- micsys.jar
- jdom.jar
- commons-logging.jar
- xercesImpl.jar
- xmlParserAPIS.jar

You can find the following files in the SDK\engine\lib\client and SDK\engine\lib\client folder under the
directory where you installed the product:

- mmjclient.jar
- mmjclient_usa.jar

Also make sure that the path to encoding-map.xml is in your classpath.

You can find this file in the SDK\engine\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 USA API.

/* 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";
Developing Geocoding Applications in Java

/* 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(canAddr);

        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 */
ext 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();
        }
    }
} /* End */
public static void main(String[] args) {
    MMJ_Generic_Example geocodeTest =
        new MMJ_Generic_Example();
    geocodeTest.geocodeGenericAddress();
}

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();
    inpAddr.setMainAddress("G");
    inpAddr.setPostCode1("12180");
    inpAddr.setCountry("USA");

For detailed information on all classes, refer to the API documentation for MapMarker USA. It is located in the SDK\tomcat\webapps\ROOT\mapmarker_usa\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 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 126.

    /* 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.

    /* Get candidate count */
    int candCount = geoRes.candidateCount();
    /*
     * Get returned candidates
### Geocoding to City Centroids

The MapMarker Java USA API enables you to geocode records to city centroids. Please note that this feature is not available in the MapMarker 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
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.

```java
AddressImpl inpAddr = new AddressImpl();
inpAddr.setAreaName3("Albany");
inpAddr.setAreaName1("NY");
inpAddr.setCountry("USA");
```

For detailed information on all classes, refer to the API documentation for MapMarker USA. It is located in the SDK\tomcat\webapps\ROOT\mapmarker_usa\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 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 126.

```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.
/*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.
/* 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 georesult 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 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 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 126.

/* 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.

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:

```
USA_CustomGeocoder.GEOCODE_TYPE_AIRPORTS_BY_STATE.
```

```
/* 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.

```
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 USA. It is located in the SDK\tomcat\webapps\ROOT\mapmarker_usa\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 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 126.

```
/* 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.

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

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

`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.

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 USA. It is located in the SDK\tomcat\webapps\ROOT\mapmarker_usa\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 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 126.

/* 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.

USA_UserAirportCandidateAddress candAddr = new
USA_UserAirportCandidateAddress(geoRes.candidateAt(0));

Address Point Interpolation

This method of interpolation assigns coordinates to address records in relation to the position of address point candidates in the data, providing greater positional accuracy to the geocoded points. The geocoding engine can incorporate address point data in the position assignment and interpolation process.

MapMarker uses address point data that may be present in the Address Dictionary, or supplied by the user in a customized dictionary. The most consistent results will be achieved by using a customized dictionary that contains address points for the area you are geocoding.
To use address point interpolation, you must set the `setAddressPointInterpolation` method in the `IGeocodeConstraints` object to True.

The following types of candidates can be returned:

- The candidate to be returned is an address point candidate. If the candidate is itself an address point candidate, the unique point from the segment will be returned and the point precision will be set to 16 (CandidateAddress.ADDRESS_POINT_PRECISION).
- The candidate to be returned is not an address point candidate, but another candidate on the same street is an address point candidate and shares the same house number (e.g., the candidate to be returned was generated from the Address Dictionary and the other candidate from a user dictionary). In this case, the unique point from the second candidate is returned and the point precision is set to 16 (CandidateAddress.ADDRESS_POINT_PRECISION).
- The candidate to be returned is not an address point candidate and there are no address point candidates with a matching house number in the list of generated candidates.

In these cases, all of the address point candidates are filtered for the following information:

- A house number that is the same odd/even as the house number of the candidate to be returned.
- A point that intersects the segment of the candidate to be returned.
- A house number that is contained within the range defined for this candidate, e.g., one of this candidate’s ranges must intersect the address point candidate’s house number.

MapMarker then uses the filter information from the resulting list of address point candidates to interpolate the candidate’s point location. The precision for this type of interpolation is 17 (ADDRESS_POINT_INTERPOLATED).

If no address point candidates are returned, the point is interpolated as it would be without using address point interpolation.
Address Point Interpolation
MapMarker Sample Application

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

In this chapter:

- Requirements and Setup ........................................ 140
- Using the Sample Application ................................. 140
Requirements and Setup

Before you try to geocode an address using the application, make sure that you have access to a MapMarker server. To start the server, run the startup.bat (Windows) or startup.sh (UNIX) from the \SDK\tomcat\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 (MapMarkerUSA Sample Application.exe) from the \SDK\examples-usa directory where you installed MapMarker. Windows users can also use the Start menu 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 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 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

You can set several preferences that modify 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 (MapMarker Plus only)

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 dialog box.
Area Geocoding

If you provide a city or town name, you can geocode to the city or town. 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. For example, you could specify the intersection of 4th St. && Broadway the Street Name field.

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.
MapMarker uses XML to communicate between clients and the MapMarker server. The MapMarker XML API allows developers access to MapMarker functionality from virtually any development platform. This chapter explains how you can use the XML API to add geocoding functionality to .NET applications.

In this chapter:

- MapMarker XML Programming in .NET Framework .............. 144
- Interacting with a MapMarker Server ............................... 144
- Using XSD Files to Create .NET Classes ........................... 144
- Example Class Modifications ................................. 152
MapMarker XML Programming in .NET Framework

MapMarker Server operates using a standard XML request/response methodology. A client application is responsible for constructing a geocoding request, which must conform to the MapMarker request schema, and sends that request to an active MapMarker server via an HTTP Post. If the request message is valid MapMarker will attempt to carry out the requested operation, and if successful, will send an XML response message, which will conform to the MapMarker response schema.

Although MapMarker does not have a native .NET API, developers can still add geocoding functionality to their .NET applications via this XML API. The sections that follow provide a general outline of a number of different methods for interacting with MapMarker in this way and provide a set of detailed instructions for using some of the tools included with the .NET framework.

Interacting with a MapMarker Server

There are several methods you can use to interact with MapMarker server using the MapMarker XML API:

1. Build XML messages from scratch using string or StringBuilder objects. This method requires the programmer to be familiar with the structure of the request and response messages by studying the message schema. The programmer will then need to write code to build a string that contains all of the XML tags as well as any data that is required for a particular geocoding request. The programmer will also be required to write code to parse out the data from the XML response. In general this method is not recommended since it would require a significant amount of custom code to efficiently build the response message and parse the request message.

2. Create XML document and node objects. In this scenario the programmer can use the .NET XML document and node classes to construct an XML document that will contain all of the required information to generate an XML message or parse the results of a response. This method will require a fair amount of code to implement but should be fairly straightforward for a programmer that is familiar with the .NET XML document and node classes. Developers interested in this approach should refer to the documentation relating to these classes in Visual Studio .NET.

3. Generate .NET classes from MapMarker request and response schemas. This scenario involves generating .NET classes that are based on the message schemas and then incorporating those classes in a project to serialize and deserialize the request and response messages. This method offers a very high degree of flexibility and is easier to implement than the other two methods but the tools and methodology are not well documented. The remainder of this chapter will provide an overview of this approach as well as provide some practical examples.

Using XSD Files to Create .NET Classes

Visual Studio .NET ships with a tool (XSD.EXE) that can be used, among other things, to generate .NET class files from an XSD file. To use the XSD tool, do the following:
Chapter 8: Using the XML API

- Start the .NET command prompt, which can be found in Program Files under the Visual Studio .NET tools.

  **Note** This will not work with the standard windows command prompt; the XSD utility must be run using the .NET command prompt.

- The syntax for viewing all of the input parameters for this tool is:
  
  ```
  xsd /?
  ```

- The syntax for generating .NET class files in C# is:
  
  ```
  xsd SchemaFileName.xsd /c
  ```

- The syntax for generating .NET class files in VB.NET is:
  
  ```
  xsd SchemaFileName.xsd /c /l:VB
  ```

The XSD tool will create a class file with the same name as the xsd input file but with a different extension, .cs for C# or .vb for VB.NET. This document uses C# for all examples. It will also overwrite an existing class file without any confirmation message. You will need to run the XSD tool against both the MMJRequest.xsd and MMJResponse.xsd schemas. Once the class files have been created they can be used with any .NET application that needs to integrate with MapMarker.

Building a Sample Application

This sample application will allow a user to enter and geocode a street. It has a country input field to enable you to indicate the country of the address you are geocoding. It is assumed that the developer can connect to a running MapMarker Server.

Setting Up the Project

To set up the project, do the following:

1. Create a new Windows project in Visual Studio .NET.

2. Copy both of the class files created with the XSD tool to the project directory and add them to the project.

3. Edit the MMJResponse.cs class file to remove duplicate class definitions. The response and request schemas share a common set of schema definitions that are stored in the MMJCommon.xsd file. The XSD tool will put these common classes in both of the class files that it creates which will cause errors when the project is compiled. The easiest way to deal with this is to attempt to build the project and then delete all of the duplicate class definitions that are identified by Visual Studio.

4. Add the System.XML reference to the .NET namespaces, which is needed for XML serialization and HTTP communication. (It may be added by default when you create the Windows project.)

   Three “Using” clauses must be added to the Form1.cs file:

   ```
   Using System.XML.Serialization;
   Using System.Net;
   Using System.IO;
   ```

   If “Using System.XML” is not already included, add it to the Form1.cs file now.
5. Add the following controls to the project’s form:

<table>
<thead>
<tr>
<th>Control Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextBox</td>
<td>txtAddress</td>
</tr>
<tr>
<td>TextBox</td>
<td>txtCity</td>
</tr>
<tr>
<td>TextBox</td>
<td>txtCountrySubDivision</td>
</tr>
<tr>
<td>TextBox</td>
<td>txtPostcode</td>
</tr>
<tr>
<td>TextBox</td>
<td>txtCountry</td>
</tr>
<tr>
<td>TextBox</td>
<td>txtURL</td>
</tr>
</tbody>
</table>

6. Add the following labels to identify the TextBox controls:

<table>
<thead>
<tr>
<th>Control Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>lblAddress</td>
</tr>
<tr>
<td>Label</td>
<td>lblCity</td>
</tr>
<tr>
<td>Label</td>
<td>lblCountrySubDivision</td>
</tr>
<tr>
<td>Label</td>
<td>lblPostcode</td>
</tr>
<tr>
<td>Label</td>
<td>lblCountry</td>
</tr>
<tr>
<td>Label</td>
<td>lblURL</td>
</tr>
</tbody>
</table>

7. Add these additional labels for the geocoding output:

<table>
<thead>
<tr>
<th>Control Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>lblOutputAddress</td>
</tr>
<tr>
<td>Label</td>
<td>lblAddOutput</td>
</tr>
</tbody>
</table>

When you are finished, the form should look like this:
Adding the Code

Once you have set up the project, you are ready to add code. To add the code to the application, do the following:

1. All of the code in the steps that follow should be placed in the btnGeocode_click event handler. Everything that is included in the btnGeocode_click event handler should be placed within a try/catch block. The catch block should include:

   this.lblAddOutput.Text = ex.GetBaseException().ToString();

2. Declare all of the objects required to geocode.

   RequestEnvelopeType requestEnvelope = new RequestEnvelopeType();
   ResponseEnvelopeType responseEnvelope = new ResponseEnvelopeType();
   GeocodeRequestType geocodeRequest = new GeocodeRequestType();
   GeocodeResponseType geocodeResponse = new GeocodeResponseType();
   InputAddressType inputAddress = new InputAddressType();
   inputAddress.Address = new AddressType();
   string xmlRequest;
   StringWriter sw = new StringWriter();
   StringReader sr;
   string xmlResponse;
   WebClient web = new WebClient();
   string mainAddress = null;
   string city = null;
   string postcode = null;
   string latitude = null;
   string longitude = null;
   string georesult = null;

3. Set the input address properties.

   The following lines of code will set the appropriate properties of the input address object.

   inputAddress.Address.MainAddress = this.txtAddress.Text;
   inputAddress.Address.AreaName3 = this.txtCity.Text;
4. Add the input address object to the geocode request object.

A single MapMarker geocode request can contain multiple input addresses that can be processed with one call to the server. The input addresses are stored in an array, which will be dealt with in more detail later in the document. For now add the following line of code to include a single input address.

`geocodeRequest.InputAddress = new InputAddressType[] {inputAddress};`

5. Instantiate the request constraints and add the geocode request to the request envelope.

The MapMarker server supports a number of different types of requests. The following lines of code will add the geocode request to the request envelope and set the request type appropriately. In addition, the geocoding constraints will be set to the default settings. Geocoding constraints will be covered in more detail later.

`geocodeRequest.GeocodeConstraints = new GeocodeConstraintsType();
geocodeRequest.GeocodeConstraints.BaseConstraints = new BaseConstraintsType();
requestEnvelope.Item = geocodeRequest;
requestEnvelope.ItemElementName = ItemChoiceType.GeocodeRequest;`

6. Serialize the request envelope object to XML

The next step is to convert the request envelope object into an XML representation, which is accomplished using the .NET XML serialization namespace. The process involves defining a default namespace for the serializer, instantiating a serializer object, serializing the request envelope to a string writer, and then extracting the contents of the string writer to a string variable, which will contain our XML. The following lines of code will carry out these steps.

`XmlSerializerNamespaces ns = new XmlSerializerNamespaces();
ns.Add("", "");

XmlSerializer serializer = new XmlSerializer(requestEnvelope.GetType());
serializer.Serialize(sw, requestEnvelope, ns);
xmlRequest = sw.ToString();`

7. Convert the XML to a Byte array.

The XML string data must be converted to a byte array to be posted to the MapMarker server, using the following line of code.

`Byte[] requestStream = System.Text.Encoding.Unicode.GetBytes(xmlRequest);`

8. Set the appropriate HTTP headers.

There are a set of HTTP headers that are associated with each MapMarker request. The following lines of code show the required headers for a geocoding request.

`web.Headers.Add("Content-Type", "text/xml");
web.Headers.Add("MI_XMLProtocolRequest", "GeocodeRequest");
web.Headers.Add("MI_XMLProtocolVersion", "MI_XML_Protocol_GeocodeRequestAndResponse_1_0");`
web.Headers.Add("MI_XMLProtocolTransactionId", "0000");

9. Post the XML request to MapMarker.

The geocoding request can now be passed to an active MapMarker server, which if successful will send back a response in XML format. The following code will carry out the operation, but the first parameter will have to be modified to point to an active MapMarker server. The data will be returned as a Byte array that can be converted to a string variable.

Byte[] responseStream = web.UploadData("this.txtURL.Text", "POST", requestStream);
xmlResponse = System.Text.Encoding.ASCII.GetString(responseStream);

10. Deserialize the response XML into a Response envelope object. The final step is to convert the response from the MapMarker server into a .NET object. This is accomplished with another object from the XML serialization namespace. After the message has been deserialized the properties of the object can be checked to determine the results of the geocoding operation.

XmlSerializer deserializer = new XmlSerializer(responseEnvelope.GetType());
sr = new StringReader(xmlResponse);
responseEnvelope = (ResponseEnvelopeType)deserializer.Deserialize(sr);

if (Convert.ToInt32(gr.RequestResult[0].GeocodeSummary.TotalLocationsFound) <= 0)
{
    // if there were no candidates found then return message and exit method
    this.lblOutputAddress.Text = "";
    this.lblAddOutput.Text = "No Candidates found";
    return;
}

GeocodeResponseType gr = (GeocodeResponseType)responseEnvelope.Item;

mainAddress = gr.RequestResult[0].Candidate[0].fmtdAddr;
city = gr.RequestResult[0].Candidate[0].Address.AreaName1;
province = gr.RequestResult[0].Candidate[0].Address.AreaName3;
postcode = gr.RequestResult[0].Candidate[0].Address.postCode1;
latitude = gr.RequestResult[0].Candidate[0].Point.coord.Y;
longitude = gr.RequestResult[0].Candidate[0].Point.coord.X;
KeyValuePairType[] addFields = gr.RequestResult[0].Candidate[0].Address.AdditionalFields;
for (int i = 0; i < addFields.Length; i++)
{
    if (addFields[i].Key == "RESULT_CODE")
        georesult = addFields[i].Value;
}
this.lblOutputAddress.Text = mainAddress +", " + city +", " + province +", " + postcode +" [" + latitude +"," + longitude +"]";
Sample Application Code

The following code sample shows what the sample application code looks like:

```csharp
{ 
    try 
    { 
        RequestEnvelopeType requestEnvelope = new RequestEnvelopeType(); 
        ResponseEnvelopeType responseEnvelope = new ResponseEnvelopeType(); 
        GeocodeRequestType geocodeRequest = new GeocodeRequestType(); 
        GeocodeResponseType geocodeResponse = new GeocodeResponseType(); 
        InputAddressType inputAddress = new InputAddressType(); 
        inputAddress.Address = new AddressType(); 
        string xmlRequest; 
        StringWriter sw = new StringWriter(); 
        StringReader sr; 
        string xmlResponse; 
        WebClient web = new WebClient(); 
        string mainAddress = null; 
        string city = null; 
        string province = null; 
        string postcode = null; 
        string latitude = null; 
        string longitude = null; 
        string georeresult = null; 
        inputAddress.Address.MainAddress = this.txtAddress.Text; 
        inputAddress.Address.AreaName3 = this.txtCity.Text; 
        inputAddress.Address.AreaName1 = this.txtCountrySubDivision.Text; 
        inputAddress.Address.postCode1 = this.txtPostcode.Text; 
        inputAddress.Address.Country = this.txtCountry.Text; 
        geocodeRequest.InputAddress = new InputAddressType[] { inputAddress }; 
        geocodeRequest.GeocodeConstraints = new GeocodeConstraintsType(); 
        geocodeRequest.GeocodeConstraints.BaseConstraints = new BaseConstraintsType(); 
        requestEnvelope.Item = geocodeRequest; 
        requestEnvelope.ItemElementName = ItemChoiceType.GeocodeRequest; 
        XmlSerializerNamespaces ns = new XmlSerializerNamespaces(); 
        ns.Add('', ''); 
    } 
} 
```
XmlSerializer serializer = new XmlSerializer(requestEnvelope.GetType());
    serializer.Serialize(sw, requestEnvelope, ns);
    xmlRequest = sw.ToString();

    byte[] requestStream = System.Text.Encoding.Unicode.GetBytes(xmlRequest);

    web.Headers.Add("Content-Type", "text/xml");
    web.Headers.Add("MI_XMLProtocolRequest", "GeocodeRequest");
    web.Headers.Add("MI_XMLProtocolVersion", "MI_XML_Protocol_GeocodeRequestAndResponse_1_0");
    web.Headers.Add("MI_XMLProtocolTransactionId", "0000");

    byte[] responseStream = web.UploadData(this.txtURL.Text, "POST", requestStream);
    xmlResponse = System.Text.Encoding.ASCII.GetString(responseStream);

    XmlSerializer deserializer = new XmlSerializer(responseEnvelope.GetType());
    sr = new StringReader(xmlResponse);
    responseEnvelope = (ResponseEnvelopeType)deserializer.Deserialize(sr);

    GeocodeResponseType gr = (GeocodeResponseType)responseEnvelope.Item;

    if (Convert.ToInt32(gr.RequestResult[0].GeocodeSummary.TotalLocationsFound) <= 0)
    {
        // if there were no candidates found then return message and exit method
        this.lblOutputAddress.Text = "";
        this.lblAddOutput.Text = "No Candidates found";
        return;
    }

    mainAddress = gr.RequestResult[0].Candidate[0].fmtdAddr;
    city = gr.RequestResult[0].Candidate[0].Address.AreaName1;
    province = gr.RequestResult[0].Candidate[0].Address.AreaName3;
    postcode = gr.RequestResult[0].Candidate[0].Address.postCode1;
    latitude = gr.RequestResult[0].Candidate[0].Point.coord.Y;
    longitude = gr.RequestResult[0].Candidate[0].Point.coord.X;
    KeyValuePairType[] addFields = gr.RequestResult[0].Candidate[0].Address.AdditionalFields;
    for (int i = 0; i < addFields.Length; i++)
    {
        if (addFields[i].Key == "RESULT_CODE")
            georesult = addFields[i].Value;
Example Class Modifications

There are many changes that can be made to the source code generated by the XSD tool that will make working with the MapMarker server more efficient. This section of the document will briefly describe a few of the more basic modifications.

Adding Objects to Collections in the Geocode Request

Since the collections within the Geocode Request class are all arrays and since it is somewhat cumbersome to add objects to arrays in C# it would be advisable to add methods to the request class to make performing these operations more efficient.

Note It is possible to replace the arrays with ArrayLists but this makes the serialization and deserialization of the objects much more complex.

The MapMarker Server supports multiple input addresses within a single request. To make this process simpler, add the following method to the GeocodeRequestType class:

```csharp
public void AddInputAddress(InputAddressType newAddress)
{
    if (InputAddress == null)
    {
        InputAddress = new InputAddressType[1];
        InputAddress[0] = newAddress;
    }
    else
    {
        InputAddressType[] newList = new InputAddressType[InputAddress.Length + 1];
        InputAddress.CopyTo(newList, 0);
        newList[InputAddress.Length] = newAddress;
        InputAddress = newList;
    }
}
```
Some of the constraints that are used to control the geocoding process are contained with the Additional Constraints class.

**Note**  A description of the additional constraints can be found in the MapMarker API documentation.

To make the process of adding constraints simpler add the following method to the GeocodeConstraintsType class:

```csharp
public void AddConstraint(string constraintName, string constraintValue)
{
    KeyValuePairType newKeyPair = new KeyValuePairType();
    newKeyPair.Key = constraintName;
    newKeyPair.Value = constraintValue;

    if (AdditionalConstraints == null)
    {
        AdditionalConstraints = new KeyValuePairType[1];
        AdditionalConstraints[0] = newKeyPair;
    }
    else
    {
        KeyValuePairType[] newList = new
        KeyValuePairType[AdditionalConstraints.Length + 1];
        AdditionalConstraints.CopyTo(newList, 0);
        newList[AdditionalConstraints.Length] = newKeyPair;
        AdditionalConstraints = newList;
    }
}
```

**Request Web Headers**

Each type of MapMarker request has its own set of HTTP headers that must be specified before sending the request to the MapMarker server. The following code snippets illustrate the required headers for each type of supported request.

**Geocode Request**

```csharp
web.Headers.Clear();
web.Headers.Add("Content-Type", "text/xml");
web.Headers.Add("MI_XMLProtocolRequest", "GeocodeRequest");
web.Headers.Add("MI_XMLProtocolVersion", "MI/XML_Protocol_GeocodeRequestAndResponse_1_0");
web.Headers.Add("MI_XMLProtocolTransactionId", "0000");
```

**Get Versions Request**

```csharp
web.Headers.Clear();
web.Headers.Add("Content-Type", "text/xml");
web.Headers.Add("MI_XMLProtocolRequest", "VersionsRequest");
```
Example Class Modifications

Get License Information Request
web.Headers.Clear();
web.Headers.Add("Content-Type", "text/xml");
web.Headers.Add("MI_XMLProtocolRequest", "LicenseInformationRequest");

Get Dictionary Search Order Request
web.Headers.Clear();
web.Headers.Add("Content-Type", "text/xml");

Get Constraints Request
web.Headers.Clear();
web.Headers.Add("Content-Type", "text/xml");
web.Headers.Add("MI_XMLProtocolRequest", "ConstraintsRequest");
Deploying Applications

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

In this chapter:

- Deploying MapMarker as a Servlet ............... 156
- Integration with MapMarker for Other Countries ............... 156
Deploying MapMarker 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 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 is installed in the Tomcat Servlet Container 5.5. 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 > 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 is verified and supported on Sun Application Server 9.0, BEA WebLogic 9.0, and WebSphere 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 (for example., 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 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 12.x 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 12.x.
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 ........................................ 160
- Using Custom Dictionaries .............................................. 164
Creating a Custom Dictionary

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

Custom user dictionaries created in MapMarker v.10.x and earlier are not compatible with MapMarker 12.x. We recommend that you re-create the dictionary in MapMarker 12.x 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 12.x format.

For more information on the User Dictionary Wizard and converting existing dictionaries to the MapMarker 12.x format, see Chapter 8: User Dictionaries in the MapMarker Desktop User Guide.

To create a user dictionary programmatically with MapMarker 12.x, 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 12.x.

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 Abbreviation</td>
<td>Two-character state abbreviation</td>
</tr>
<tr>
<td>Left ZIP Code</td>
<td>ZIP Code for left side of street</td>
</tr>
<tr>
<td>Right ZIP Code</td>
<td>ZIP Code for right side of the street</td>
</tr>
</tbody>
</table>
Chapter 10: Creating and Using Custom Address Dictionaries

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

* These fields are highly recommended.
GeoEngDDCreateUserDictionary()

If your data includes placenames, the dictionary contains the following files:

- udl.pdx
- udl.pbx

The dictionary also contains these log files:

- udl.log
- udl.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 \texttt{GeoEngDDCreateUserDictionary()}, datadict.dll must be included in your list of libraries.

GeoEngDDCreateUserDictionary()

**Purpose**

This function create a user-defined address dictionary from a MapInfo table to be used in geocoding with MapMarker’s GeoEngine.

**Syntax**

\begin{verbatim}
long GeoEngDDCreateUserDictionary(GEO_ENG_HANDLE geoEngHandle,
       char *input_table_name,
       char *output_dictionary_name,
       pUD_INFO   pUDInfo
       (long (*pStatusFunction)(long)))
\end{verbatim}

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoEngHandle</td>
<td>An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to GeoEngInit().</td>
</tr>
<tr>
<td>input_table_name</td>
<td>Full path of the MapInfo input table name from which a user dictionary is created.</td>
</tr>
<tr>
<td>output_dictionary_name</td>
<td>Full path to the output dictionary including the user dictionary file extension .udr.</td>
</tr>
</tbody>
</table>
This API routine creates a user dictionary from a MapInfo input table. The user dictionary is named according to the output_dictionary_name argument given to the routine. The GeoEngine handle is used to look up ZIP Codes in the 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 display the status on 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];
} pUDInfo
```

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUDInfo</td>
<td>Pointer to the UD_Info (user dictionary information) structure</td>
</tr>
<tr>
<td>pStatusFunction</td>
<td>Pointer to status function</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUDInfo</td>
<td>Pointer to the UD_Info (user dictionary information) structure</td>
</tr>
<tr>
<td>pStatusFunction</td>
<td>Pointer to status function</td>
</tr>
</tbody>
</table>
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 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.

To specify the order of the configured dictionaries, change the rank of each dictionary in the corresponding path key as necessary. When there are close match candidates from different dictionaries, the close matches that are returned come from the dictionary that is ranked first in the search order. Close matches from lower ranked dictionaries are demoted to non-close matches.

Specifying Preferences Using the MapMarker Java API

Users of a server product can specify the order of dictionaries and 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 specific 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
- user search order
- 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.

For example, clients may call GetDictionarySearchOrder, then in the constraints make changes to the search order, and set the resulting object in the constraints using the setDictionarySearchOrder method. For example, if the server has three configured dictionaries, the client can specify that they want candidates from a particular dictionary number.
Some of the methods that are used with GetDictionarySearchOrder are described briefly in the following table:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDictionaryCount</td>
<td>Returns the number of configured dictionaries available to the engine.</td>
</tr>
<tr>
<td>getDictionaryDescription</td>
<td>Returns the description of the indicated dictionary. If no description is found, a string indicating whether the dictionary is a user dictionary will be returned. To create a dictionary description, type the description text into a text file and name it &quot;dictionarydesc.txt.&quot; Then place the file in the dictionary path.</td>
</tr>
<tr>
<td>getSearchOrderForDictionary</td>
<td>Returns the dictionary search order.</td>
</tr>
<tr>
<td>isDictionaryAvailableForSearch</td>
<td>Returns true if dictionary will be searched.</td>
</tr>
<tr>
<td>isUserDictionary</td>
<td>Returns true if the dictionary is a user-created dictionary.</td>
</tr>
<tr>
<td>setDictionaryAvailableForSearch</td>
<td>Setting to false indicates that the dictionary will not be searched.</td>
</tr>
<tr>
<td>setSearchOrderForDictionary</td>
<td>Sets the order in which the dictionaries will be searched.</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 setSearchOrderForDictionary 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 USA Java API" in the box labeled MapMarker Documentation.
Chapter 10: Creating and Using Custom Address Dictionaries

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 UDs.</td>
</tr>
<tr>
<td>PREFER_AD</td>
<td>Use both address dictionaries (AD) and user dictionaries (UD) but give preference to candidates from ADs.</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.

In this appendix:

- J Server 3.1 to MapMarker Java 12.x Conversion . . . . . . . . . . . . . . . . . . 170
J Server 3.1 to MapMarker Java 12.x Conversion

This appendix provides an example of converting a MapMarker J Server 3.1 application to MapMarker Java 12.x.

Text Conventions

The following text conventions have been used in this example to illustrate what has been added for MapMarker 12.x, and what has been deprecated.

- Bold text indicates changes or additions for MapMarker 12.x.
- 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 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 12 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 12.0 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 12, 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()
    .copy(geoRes.candidateAt(i));

    /* MM 12.0 changes */
    USA_UserCandidateAddress candAddr = new USA_UserCandidateAddress(geoRes.candidateAt(i));

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

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

    /* MM 12 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 12 changes*/
    System.out.println("Street:" +
        candAddr.getGeoResult());
    System.out.println("Census:" +
        candAddr.getCensusBlock());

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

} // end try
/* MM 12 exception handling */
catch (MapMarkerFatalException e)
{
    System.out.println("Fatal Exception:" + e.getMessage());
}
catch (MapMarkerException e)
{
    System.out.println("Exception:" + e.getMessage());
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 ................................................................. 176
- Country and Coordinate System Settings ....................... 176
JVM Settings

The configuration file controls the creation of the JVM in which the Java engine runs.

The following JVM settings are useful for changing the amount of memory used by the JVM.

+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 provides a reference to error messages you may receive in the course of using MapMarker.

In this appendix:

- MapMarker Java API Errors, p. 178
- Geocoding Engine Errors, p. 182
- Adapter Errors, p. 183
# MapMarker Java API Errors

## Engine Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR_CANT_CREATE_GEOCODABLE_ADDRESS</td>
<td>Engine unable to create an address object for the given country.</td>
</tr>
<tr>
<td>ERR_EXCEPTION_IN_ENGINE</td>
<td>Engine encountered an exception.</td>
</tr>
<tr>
<td>ERR_UNKNOWN_GEOCODE_TYPE</td>
<td>Engine was given an unknown geocode type.</td>
</tr>
<tr>
<td>ERR_ENGINE_RESET_NULL</td>
<td>Engine returned null instead of a response object.</td>
</tr>
</tbody>
</table>

## Parser Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_PARSER_INPUT_ADDRESS_NULL</td>
<td>2000</td>
<td>Input address is null.</td>
</tr>
<tr>
<td>ERROR_PARSER_PARSED_ADDRESS_NULL</td>
<td>2001</td>
<td>Parsed address is null.</td>
</tr>
<tr>
<td>ERROR_PARSER_INITIALIZATION</td>
<td>5000</td>
<td>Unable to initialize parser.</td>
</tr>
</tbody>
</table>

## Data Access Exceptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_DATA_ACCESS_INVALID_REQUEST</td>
<td>2100</td>
<td>Invalid request made during data access.</td>
</tr>
<tr>
<td>ERROR_DATA_ACCESS_DATA_CORRUPTED</td>
<td>2101</td>
<td>Data corruption encountered during data access.</td>
</tr>
<tr>
<td>ERROR_DATA_ACCESS_IO</td>
<td>2102</td>
<td>IO error encountered during data access.</td>
</tr>
<tr>
<td>ERROR_DATA_ACCESS_NO_DICTIONARY</td>
<td>2103</td>
<td>The data manager failed to find any usable dictionaries, based on the system preferences and settings.</td>
</tr>
</tbody>
</table>
Chapter C: Error Messages

### General Geocoding Process Exceptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_GAC_RESOURCE</td>
<td>2300</td>
<td>Unable to find resource bundle for error messages.</td>
</tr>
<tr>
<td>ERROR_GAC_FACTORY_INSTANTIATION</td>
<td>2301</td>
<td>Unable to instantiate geocodable address factory for country code {0}.</td>
</tr>
<tr>
<td>ERROR_NULL_RESULT</td>
<td>2302</td>
<td>The geocoding engine was unable to return a result.</td>
</tr>
<tr>
<td>ERROR_GEO_TYPE_UNSUPPORTED</td>
<td>2303</td>
<td>Unsupported geocode type: {0}</td>
</tr>
<tr>
<td>ERROR_NOPARSE_RESULT</td>
<td>2304</td>
<td>No result received from parser.</td>
</tr>
<tr>
<td>ERROR_GAC_PARSER_INIT_FAILED</td>
<td>2305</td>
<td>Error initializing parser.</td>
</tr>
</tbody>
</table>

### License Exceptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_LICENSE_INIT</td>
<td>2200</td>
<td>Unable to initialize license manager.</td>
</tr>
<tr>
<td>ERROR_LICENSE_MISSING_BASIC_LICENSE</td>
<td>5200</td>
<td>Missing required initialization license.</td>
</tr>
<tr>
<td>ERROR_LICENSE_ILLEGAL_GEOCODE_MODE</td>
<td>5201</td>
<td>Illegal geocode mode encountered during license evaluation.</td>
</tr>
</tbody>
</table>

### Error Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_DATA_ACCESS_NO_SAC</td>
<td>2104 No search area codes found for given location information.</td>
</tr>
<tr>
<td>ERROR_DATA_ACCESS_NO_STREET_BASE</td>
<td>2105 No street base available for given street name.</td>
</tr>
<tr>
<td>ERROR_DATA_ACCESS_NO_SAC_DATA</td>
<td>2106 Data construction is not optimized. No data files found for given search area code: {0}.</td>
</tr>
<tr>
<td>ERROR_DATA_ACCESS_MISSING_FILE</td>
<td>2107 Data file missing.</td>
</tr>
<tr>
<td>ERROR_DATA_ACCESS_DATA_NOT_LICENSED</td>
<td>2108 Data not licensed.</td>
</tr>
<tr>
<td>ERROR_DATA_ACCESS_MISSING_FILE</td>
<td>5100 Missing file: {0}.</td>
</tr>
</tbody>
</table>
### MapMarker Java API Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_GAC_DATA_MANAGER_INIT_FAILED</td>
<td>2306</td>
<td>Error initializing data manager.</td>
</tr>
<tr>
<td>ERROR_NOIMPLEMENTATION</td>
<td>2307</td>
<td>Engine method not implemented.</td>
</tr>
<tr>
<td>ERROR_UNEXPECTED_ERROR</td>
<td>2308</td>
<td>Unexpected error encountered.</td>
</tr>
<tr>
<td>ERROR_BATCH_SIZE_EXCEEDED</td>
<td>2309</td>
<td>Maximum batch size exceeded.</td>
</tr>
<tr>
<td>ERROR_UNSUPPORTED_XML_PROTOCOL</td>
<td>2310</td>
<td>Unsupported XML Protocol request.</td>
</tr>
</tbody>
</table>

### Client Exceptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_CLIENT_BAD_URL</td>
<td>2400</td>
<td>Malformed URL.</td>
</tr>
<tr>
<td>ERROR_CLIENT_HTTP_ERROR</td>
<td>2401</td>
<td>HTTP Error: {0}.</td>
</tr>
<tr>
<td>ERROR_CLIENT_JDOM</td>
<td>2402</td>
<td>JDom Exception encountered.</td>
</tr>
<tr>
<td>ERROR_CLIENT_XML_TRANSFORM</td>
<td>2403</td>
<td>Error during XML transformation.</td>
</tr>
<tr>
<td>ERROR_CLIENT_IO</td>
<td>2404</td>
<td>IO Error.</td>
</tr>
<tr>
<td>ERROR_CLIENT_MISSING_URL</td>
<td>2405</td>
<td>The URL to the servlet has not been set. Please try again with a Servlet URL.</td>
</tr>
<tr>
<td>ERROR_CLIENT_MISSING_RESPONSE</td>
<td>2406</td>
<td>No response was returned from server.</td>
</tr>
</tbody>
</table>

### Localization (Geocoder Construction Time) Exceptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_HANDLER_INVALID</td>
<td>2500</td>
<td>The provided IHandler implementation is not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INCOMPLETE_NO_DATA_MANAGER</td>
<td>2501</td>
<td>The provided IHandler implementation is not complete. No local data manager was provided.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Error #</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ERROR_HANDLER_INCOMPLETE_NO_PARSER_RULES</td>
<td>2502</td>
<td>The provided IHandler implementation is not complete. No local address parsing rules were provided.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_ADDRESS_GEOCODER</td>
<td>2503</td>
<td>The IAddressGeocoder provided in the IHandler implementation is required but not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_POSTAL_GEOCODER</td>
<td>2504</td>
<td>The IPostalGeocoder provided in the IHandler implementation is required but not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_GEOGRAPHIC_GEOCODER</td>
<td>2505</td>
<td>The IGeographicGeocoder provided in the IHandler implementation is required but not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_BROWSE_GEOCODER</td>
<td>2506</td>
<td>The IBrowseGeocoder provided in the IHandler implementation is required but not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_STANDARDIZE_GEOCODER</td>
<td>2507</td>
<td>The IStandardizeGeocoder provided in the IHandler implementation is required but not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_CUSTOM_GEOCODER</td>
<td>2508</td>
<td>The ICustomGeocoder provided in the IHandler implementation is required but not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_POSTAL_FILTER</td>
<td>2509</td>
<td>The IPostalCentroidFilter provided in the IHandler implementation is not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_GEOGRAPHIC_FILTER</td>
<td>2510</td>
<td>The IGeographicCentroidFilter provided in the IHandler implementation is not valid.</td>
</tr>
<tr>
<td>ERROR_HANDLER_INVALID_PARSER</td>
<td>2511</td>
<td>The parser provided in the IHandler implementation is not valid.</td>
</tr>
</tbody>
</table>
### Server Exceptions

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR_SERVER_JDOM</td>
<td>2600</td>
<td>The server was unable to process the request.</td>
</tr>
<tr>
<td>ERROR_SERVER_INVALID_REQUEST_TYPE</td>
<td>2601</td>
<td>An invalid request type was provided.</td>
</tr>
</tbody>
</table>

### Geocoding Engine Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO_ENG_MALLOC_ERR</td>
<td>1</td>
<td>The engine was unable to allocate memory.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_PARAM_ERR</td>
<td>3</td>
<td>The engine encountered a parameter that was not specified correctly.</td>
</tr>
<tr>
<td>GEO_ENG_END_OF_DATA</td>
<td>11</td>
<td>The engine reached the end of the data.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_INPUT_ADDRESS</td>
<td>14</td>
<td>The engine encountered a bad input address.</td>
</tr>
<tr>
<td>GEO_ENG_NO_DATA_AVAILABLE</td>
<td>15</td>
<td>No data is available or engine cannot find the data.</td>
</tr>
<tr>
<td>GEO_ENG_COORDS_NOT_AVAILABLE</td>
<td>51</td>
<td>No coordinates are available.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_CAND_INDEX</td>
<td>52</td>
<td>Candidate index is incorrect.</td>
</tr>
<tr>
<td>GEO_ENG_BAD_RANGE_INDEX</td>
<td>53</td>
<td>Range index is incorrect.</td>
</tr>
</tbody>
</table>
## Adapter Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAPTER_NO_CONFIG</td>
<td>1793</td>
<td>Could not find or open the config file.</td>
</tr>
<tr>
<td>ADAPTER_EXCEEDED_OPTIONS</td>
<td>1794</td>
<td>Number of options exceeds number of options specified in config file.</td>
</tr>
<tr>
<td>ADAPTER_NO_ENGINE_CLASS</td>
<td>1797</td>
<td>Unable to find mmjni.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_NO_HELPER_CLASS</td>
<td>1801</td>
<td>Unable to find mmjni.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_GEOCODE_RESULT_NO_CLASS</td>
<td>1804</td>
<td>Unable to find mmjni.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_CANDIDATE_NO_CLASS</td>
<td>1806</td>
<td>Unable to find mmjni.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_RANGE_NO_CLASS</td>
<td>1808</td>
<td>Unable to find mmjni.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_JNI_CFG_PARMS_NO_CLASS</td>
<td>1810</td>
<td>Unable to find mmjni.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_DOUBLE_POINT_NO_CLASS</td>
<td>1814</td>
<td>Unable to find miutil.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_MAPMARKER_EXCEPTION_NO_CLASS</td>
<td>1820</td>
<td>Unable to find mmj.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_MAPMARKER_FATAL_EXCEPTION_NO_CLASS</td>
<td>1822</td>
<td>Unable to find mmj.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_NO_CONSTRAINTS_NO_CLASS</td>
<td>1824</td>
<td>Unable to find mmj.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_NO_MM_EXCEPTION_CODES_CLASS</td>
<td>1826</td>
<td>Unable to find mmj.jar in classpath.</td>
</tr>
<tr>
<td>ADAPTER_RUNTIME_ERR</td>
<td>1841</td>
<td>Runtime error occurred.</td>
</tr>
<tr>
<td>ADAPTER_OUT_OF_MEMORY</td>
<td>1842</td>
<td>Memory exceeded. Check memory specified in config file.</td>
</tr>
<tr>
<td>ADAPTER_RUNTIME</td>
<td>1843</td>
<td>Runtime exception occurred.</td>
</tr>
</tbody>
</table>
## Adapter Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAPTER_MAPMARKER_FATAL_UNKNOWN</td>
<td>1844</td>
<td>Unknown fatal exception occurred.</td>
</tr>
<tr>
<td>ADAPTER_UNKNOWN_EXCEPTION</td>
<td>1845</td>
<td>Unknown exception occurred.</td>
</tr>
<tr>
<td>ADAPTER_NO_JVM_DLL</td>
<td>1871</td>
<td>Unable to find jvm.</td>
</tr>
<tr>
<td>ADAPTER_NO_JVM_CREATE</td>
<td>1872</td>
<td>Unable to create jvm. Check jvm installation, or config file options.</td>
</tr>
</tbody>
</table>
Index

A
adapter configuration file 36, 176
address dictionary
definition 42
installing 20
storing data 23–24
storing on hard drive 23–24
address matching 10
address point interpolation 136
addresses
browsing 131
input 45–46
result code information 49
airport geocoding 134–135

B
browsing addresses 131
building .NET sample application 145–152

C
CAddressList 105
candidates
definition 42
retrieving 125
C-based developer tools 52–57
city centroid geocoding 132–133
ClearDialogText() 68
client installation 31–32
client/server geocoding 54
client/server toolkit
client/server geocoding 54
MapMarker server 55
close match
definition 42
compiling C applications 57–58
Connect() 69–70
constraints 122–124
coordinate systems
JNI adapter configuration file settings 176
setting in applications 124
country settings 176
creating
user dictionaries 160–163
Web client 126–130
creating .NET classes 144
custom geocoding control 60
customer service 18

d
data
installing 20
storing 23–24
data dictionary API 162–163
database development tools 53
database drivers 35
deployment as servlet 156
dictionary ordering 165
dictionary result information 49
Disconnect() 70
documentation set 12–13
DoGeocode() 70
DoSetProperties() 70
DpvGeocodeAddress() 71–72
DpvGeocodeAddressLastLine() 72–74
DpvGeocodeAddressLastLineWithSerial() 74–75
DpvGeocodeAddressWithSerial() 75–76

e
Envinsa 41
error message reference 178–184
ESP SQL Server 54

G
generic preferences 46
geocode requests
adding objects to collections in XML API 152–153
contents of 48
HTTP header 153
geocode types 44, 49
GeocodeAddress() 77, 80
GeocodeAddressEx() 80–81
GeocodeAddressLastLine() 81–82
GeocodeAddressLastLineEx() 83
GeocodeAddressLastLineWithSerial() 84
GeocodeAddressWithSerial() 85–86
GeocodeAddressWithSerialEx() 86
GeocodeCheckDbAvailability() 88
GeocodeFreeSet() 88
GeocodeGetCandidates() 89–90
GeocodeGetErrorText() 90
GeocodeGetHwyExitCandidate() 90
GeocodeGetServerVersion() 92
GeocodeGetStatesFound() 92
GeocodeGetStatesLicensed() 93
GeocodeHwyExit() 93
GeocodeIsCandidateMultiUnit() 95
Index

GeocodePostalCentroid() 95–96
GeocodePostalCentroidWithSerial() 96–97

gecoding
  airports 134–135
  city centroids 132–133
  client/server 54
  defined 10
  geocode method 125
  highway exits 133
  model 42–43
  preferences 42
  result codes 48–49

gecoding applications
  choosing development tool 40–41
  XML programming in .NET 144–154

Geocoding Cartridge 54

gecoding constraints 122–124

gecoding from remote workstations 55

gecoding terms 42

GeoEngDDCreateUserDictionary() 162–164

GetCandidateAt() 97
GetCandidateCensusBlockIDAt() 98
GetCandidateCityAt() 98
GetCandidateFirmAt() 99
GetCandidateLatitudeAt() 99
GetCandidateLongitudeAt() 100
GetCandidatePlus4At() 100
GetCandidatePrecisionAt() 101
GetCandidateResultCodeAt() 101
GetCandidateStateAt() 102
GetCandidateStreetAt() 102
GetCandidateZIPAt() 103
GetFullName() 103
GetName() 104
GetVersion() 104

H

highway exit gecoding 133

HTTP headers 148, 153–154

I

Informix DataBlades 54

input addresses
  building 45–46
  setting 121

installation
  client 31–32
  re-installing 33–34
  shared 31
  silent install 34
  system recommendations 24
  system requirements 20, 22
  UNIX/Linux procedure 34–35
  upgrading 20, 33
  Windows procedure 24, 26, 30–31

integrating multiple country geocoders 156–157

J

J Server
  converting application to MapMarker Plus 170–173
  deprecation status 52

Java Virtual Machine (JVM)
  adapter settings 176
  supported versions 21

javadoc location 120

JNI adapter
  configuration file settings 36, 176
  OLE automation 60
  running C applications 57–58

M

Mapinfo ftp site 17
MapInfo Professional integration 14
MapMarker J Server
  converting application to MapMarker Plus 170–173
  deprecation status 52

MapMarker OCX 54

MapMarker Plus
  documentation set 12–13
  sample application 140–142
  technical support 15–17
  what’s new 11

MapMarker requests
  HTTP headers 148, 153–154

MapMarker server
  client/server gecoding 54
  gecoding from remote workstations 55
  request HTTP headers 153–154
  request time outs 57
  running on Windows 55
  with XML API 144–154

MapMarker Streets
  using with MapMarker 14

MapXtreme 41

multiple country integration 156–157

N

NET class creation 144
NET sample application 145–152
network installation 31–32
new features and enhancements 11
Index

O

OLE automation API
- creating a custom control 60
- flow diagram 60
- list of OCX methods 66–68
- programming example 107–117
- summary of properties, events, methods 61

OLE automation methods
- ClearDialogText() 68
- Connect() 69–70
- Disconnect() 70
- DoGeocode() 70
- DoSetProperties() 70
- DpvGeocodeAddress() 71–72
- DpvGeocodeAddressLastLine() 72–74
- DpvGeocodeAddressLastLineWithSerial() 74–75
- DpvGeocodeAddressWithSerial() 75–76
- GeocodeAddress() 77, 80
- GeocodeAddressEx() 80–81
- GeocodeAddressLastLine() 81–82
- GeocodeAddressLastLineEx() 83
- GeocodeAddressLastLineWithSerial() 84
- GeocodeAddressWithSerial() 85–86
- GeocodeAddressWithSerialEx() 86
- GeocodeCheckDbAvailability() 88
- GeocodeFreeSet() 88
- GeocodeGetCandidates 89–90
- GeocodeGetErrorText() 90
- GeocodeGetHwyExitCandidate() 90
- GeocodeGetServerVersion() 92
- GeocodeGetStatesFound() 92
- GeocodeGetStatesLicensed() 93
- GeocodeHwyExit() 93
- GeocodeIsCandidateMultiUnit() 95
- GeocodePostalCentroid() 95–96
- GeocodePostalCentroidWithSerial() 96–97
- GetCandidateAt() 97
- GetCandidateCensusBlockIDAt() 98
- GetCandidateCityAt() 98
- GetCandidateFirmAt() 99
- GetCandidateLatitudeAt() 99
- GetCandidateLongitudeAt() 100
- GetCandidatePlus4At() 100
- GetCandidatePrecisionAt() 101
- GetCandidateResultCodeAt() 101
- GetCandidateStateAt() 102
- GetCandidateStreetAt() 102
- GetCandidateZIPAt() 103
- GetFullName() 103
- GetName() 104
- GetVersion() 104
- RefreshDialog() 104

SelectCandidateAt() 104

OLE automation objects 105–107

operating systems 20

Oracle Geocoding Cartridge 54

P

performance
- increasing 24

positional accuracy of points
- result code information 49

postal geocoding 44

preferences
- generic and U.S. 46
- geocoding 42
- usage examples 47
- user dictionary 165, 167

R

RefreshDialog() 104

re-installing MapMarker 33–34

remote database drivers 35

result codes
- definition 42
- interpreting 48–49
- retrieving 126

running C applications 57–58

S

sample application 140–142

SelectCandidateAt() 104

sending geocode requests 48

servlet deployment 156

setting
- geocoding constraints 122–124
- input address 121

silent install 34

starting MapMarker 36

street-level geocoding 44

Sun Application Server 9.0 21, 156

support
- customer service 18
- technical support 15–17

system recommendations 24

system requirements 20, 22

T

technical support 15–17

Tomcat Servlet Container 156

U

UNIX/Linux installation 34–35
Index

upgrading your installation 20, 33
user dictionaries
  configuring 164
  creating 160–163
  definition 42
  preferences 165, 167

W

war files 156
Web environments 21, 156
Web geocoding client 126–130
WebLogic 9.0 21, 156
WebSphere 6.0 21, 156
Windows console application
  running MapMarker server as 56
Windows service
  running MapMarker server as 55–56

X

XML API 144–154
XSD files
  creating .NET classes 144–145