



# MapMarker

Version 14

**DEVELOPER GUIDE**

Information in this document is subject to change without notice and does not represent a commitment on the part of the vendor or its representatives. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, without the written permission of Pitney Bowes Software Inc., One Global View, Troy, New York 12180-8399.

© 2008 Pitney Bowes Software Inc. All rights reserved. MapInfo, the Pitney Bowes MapInfo logo, and MapMarker are trademarks of Pitney Bowes Software Inc. and/or its affiliates.

Americas:

Phone: (518) 285-6000

Fax: (518) 285-6070

Sales: (800) 327-8627

Government Sales: (800) 619-2333

Technical Support: (518) 285-7283

Technical Support Fax: (518) 285-6080

[www.mapinfo.com](http://www.mapinfo.com)

Contact information for all Pitney Bowes Software Inc. offices is located at: <http://www.mapinfo.com/contactus>.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

The following trademarks are owned by the United States Postal Service: LACS<sup>Link</sup>, DPV, ZIP + 4, ZIP Code, ZIP, CASS, CASS Certified, Post Office, P.O. Box, USPS, U.S. Postal Service and United States Postal Service.

Pitney Bowes Software Inc. is a non-exclusive DPV and LACS<sup>Link</sup> Licensee of the United States Postal Service<sup>®</sup> AD#5.07

Portions © 2008 Tele Atlas, Inc.

Products named herein may be trademarks of their respective manufacturers and are hereby recognized. Trademarked names are used editorially, to the benefit of the trademark owner, with no intent to infringe on the trademark.

July 2008

---

# Table of Contents

---

<b>Chapter 1: Introduction</b> .....	<b>9</b>
<b>What Is Geocoding?</b> .....	<b>10</b>
<b>What Is MapMarker?</b> .....	<b>10</b>
MapMarker Desktop Application .....	10
Beyond the Desktop .....	11
<b>MapMarker Standard or MapMarker Plus</b> .....	<b>11</b>
<b>New in This Release</b> .....	<b>11</b>
ParcelPrecision™ User Dictionary .....	11
Address Dictionary Update .....	12
MapInfo Professional Integration .....	12
<b>MapMarker Documentation Set</b> .....	<b>12</b>
MapMarker User Guide .....	12
MapMarker Release Notes .....	13
Publications on the Web .....	13
Online Help .....	13
<b>MapMarker Streets</b> .....	<b>14</b>
<b>Getting Technical Support</b> .....	<b>14</b>
Before You Call .....	15
The Support Tracking System .....	15
Expected Response Time .....	15
Exchanging Information .....	16
Accessing the Pitney Bowes MapInfo FTP site .....	16
Software Defects .....	16
Customer Service (Non-Technical Support) .....	16
<b>Chapter 2: Installing MapMarker</b> .....	<b>17</b>
<b>Installation Overview</b> .....	<b>18</b>
Shared Install .....	18
About the Address Dictionary .....	19
<b>Operating Requirements and Performance Recommendations</b> .....	<b>19</b>
Operating Requirements .....	19
Hardware and Memory Requirements .....	19
Supported Operating Systems .....	20
JVM Requirements .....	20
Database Requirements .....	20
Web Server Requirements (Server Product Only) .....	20
Optimizing Performance .....	21
<b>Installing MapMarker</b> .....	<b>21</b>
Install Using the Graphical Interface .....	21

---

Installing Silently . . . . .	26
<b>Modifying the MapMarker Installation . . . . .</b>	<b>27</b>
Modify Using the Graphical Interface . . . . .	27
Modify Silently . . . . .	28
<b>Upgrading the MapMarker Installation . . . . .</b>	<b>28</b>
Upgrade Using the Graphical Interface . . . . .	29
Upgrade Silently . . . . .	29
<b>Uninstalling MapMarker . . . . .</b>	<b>29</b>
Uninstall Using the Graphical Interface . . . . .	30
Uninstall Silently . . . . .	30
<b>Client Workstation Install . . . . .</b>	<b>30</b>
<b>Starting MapMarker . . . . .</b>	<b>31</b>
Starting the Desktop Application . . . . .	31
Starting the MapMarker Server . . . . .	31
<b>Chapter 3: Designing Geocoding Applications . . . . .</b>	<b>33</b>
<b>Choosing a Development Tool . . . . .</b>	<b>34</b>
Geocoding Interoperability . . . . .	34
MapMarker Plus 14.0 . . . . .	35
MapXtreme . . . . .	35
Envinsa . . . . .	35
<b>Geocoding Terms . . . . .</b>	<b>36</b>
<b>Defining the Geocoding Model . . . . .</b>	<b>39</b>
Geocoding Preferences . . . . .	39
Use of Geocoded Data . . . . .	40
Executing the Application . . . . .	40
<b>Geocoding Application Design Overview . . . . .</b>	<b>40</b>
Types of Geocoding . . . . .	41
Street Geocoding . . . . .	41
Postal Geocoding . . . . .	41
Geographic Geocoding . . . . .	42
Building an Input Address . . . . .	42
Using Preferences . . . . .	44
Preference Usage Examples . . . . .	44
Sending a Request . . . . .	45
Verifying a Response . . . . .	45
Using Geocode Result Information . . . . .	46
Geocode Type . . . . .	46
Positional Accuracy . . . . .	46
Address Elements . . . . .	47
Dictionary Type . . . . .	47
Unmatched Elements in Input Address . . . . .	47
Optimizing DPV and LACS <sup>Link</sup> Performance . . . . .	47

---

<b>Chapter 4: Using C-Based Developer Tools</b> .....	<b>49</b>
<b>MapMarker Geocoding API Development</b> .....	<b>50</b>
Java Developers .....	50
Windows Developers .....	51
Solaris, HP-UX, and Linux Developers .....	51
Database Developers .....	52
<b>Client/Server Geocoding</b> .....	<b>52</b>
Installing MapMarker Server Manually .....	52
Registering the OCX Manually .....	53
<b>MapMarker Server</b> .....	<b>53</b>
<b>Running MapMarker Server on Windows Systems</b> .....	<b>53</b>
Configuring as a Windows Service .....	53
Configuring as a Console Application .....	54
Geocoding Request Timed Out .....	55
<b>Compiling and Running Existing Applications</b> .....	<b>55</b>
Windows Applications .....	56
Other Configuration File Settings .....	56
<b>Chapter 5: OLE Automation API</b> .....	<b>57</b>
<b>Creating a Custom Geocoding Control</b> .....	<b>58</b>
<b>Geocoder Control Properties, Events, and Methods</b> .....	<b>59</b>
<b>OLE Automation Methods</b> .....	<b>64</b>
<b>OLE Automation Objects</b> .....	<b>103</b>
CAddressList .....	103
CAddress .....	103
Footnote Codes .....	105
<b>Programming Usage Example</b> .....	<b>105</b>
<b>Chapter 6: Using the MapMarker Java API</b> .....	<b>117</b>
<b>Using the MapMarker Java USA API</b> .....	<b>118</b>
<b>Using the MapMarker Generic Java API</b> .....	<b>118</b>
<b>Using the JavaDocs API Documentation</b> .....	<b>118</b>
<b>Developing Geocoding Applications in Java</b> .....	<b>118</b>
Setting the Input Address .....	119
Geocoding Constraints .....	120
Using Match Mode Constants .....	122
USA_GeocodeConstraints .....	122
Sample Code for Setting Geocoding Constraints .....	123
Setting the Coordinate System .....	124
Geocoding .....	124
Getting Candidates .....	125
Getting the Result Code .....	126

---

<b>Creating a Web Geocoding Client in Java</b> .....	<b>126</b>
Sample Code for Web Client .....	127
<b>Browsing Addresses</b> .....	<b>131</b>
<b>Geocoding to Geographic Centroids</b> .....	<b>132</b>
<b>Geocoding to Highway Exits</b> .....	<b>134</b>
<b>Geocoding to Airports</b> .....	<b>135</b>
Finding Airports by State .....	135
Finding Airports by Code .....	136
<b>Address Point Interpolation</b> .....	<b>136</b>
<b>Determining DPV Availability</b> .....	<b>137</b>
<b>Chapter 7: MapMarker Sample Application</b> .....	<b>141</b>
<b>Requirements and Setup</b> .....	<b>142</b>
<b>Using the Sample Application</b> .....	<b>142</b>
Testing the Server .....	143
Setting Preferences .....	143
Setting Must Match Options .....	143
Postal Geocoding .....	143
Area Geocoding .....	144
Address Geocoding .....	144
Viewing Candidates .....	144
Candidate Statistics .....	144
<b>Chapter 8: Using the XML API in .NET Framework</b> .....	<b>145</b>
<b>MapMarker XML Programming in .NET Framework</b> .....	<b>146</b>
<b>Interacting with a MapMarker Server</b> .....	<b>146</b>
<b>Using XSD Files to Create .NET Classes</b> .....	<b>147</b>
<b>Tips for Adding Code for Your Application</b> .....	<b>148</b>
Request Web Headers .....	150
Geocode Request .....	150
Get Versions Request .....	151
Get License Information Request .....	151
Get Dictionary Search Order Request .....	151
Get Constraints Request .....	151
<b>Examples of XML Requests and Responses</b> .....	<b>151</b>
Street Request and Response .....	151
Postal Request and Response .....	155
Highway Exit Request and Response .....	157
Setting the Coordinate System .....	161
<b>Performance Tips</b> .....	<b>161</b>
<b>Summary of the MapMarker XML API</b> .....	<b>161</b>
CandidateAddress Class .....	162
CandidateRange Class .....	163

---

Geocoding Constraints . . . . .	164
Candidate Additional Field Values . . . . .	169
<b>Chapter 9: Deploying Applications . . . . .</b>	<b>171</b>
<b>Deploying MapMarker as a Servlet . . . . .</b>	<b>172</b>
Using the Default Apache Tomcat Servlet Container . . . . .	172
Deploying MapMarker Server in Other Web Environments . . . . .	172
<b>Integration with MapMarker for Other Countries . . . . .</b>	<b>172</b>
Running One Tomcat Web Server for Each Country . . . . .	173
Integrating Multiple Countries in One Servlet Context . . . . .	173
<b>Chapter 10: Custom User Dictionaries . . . . .</b>	<b>175</b>
<b>What Is a Custom User Dictionary? . . . . .</b>	<b>176</b>
<b>User Dictionary Capabilities and Requirements . . . . .</b>	<b>176</b>
Source Data Requirements . . . . .	176
Required Input Fields . . . . .	177
Optional (Recommended) Input Fields . . . . .	177
<b>User Dictionary File Names and Formats . . . . .</b>	<b>178</b>
<b>Additional User Dictionary Considerations . . . . .</b>	<b>179</b>
Data Access License . . . . .	179
CASS Standards . . . . .	179
Address Range Order . . . . .	180
Street Intersections and Customized User Dictionaries . . . . .	180
<b>Creating a Custom User Dictionary . . . . .</b>	<b>181</b>
Using the MapMarker User Dictionary Wizard . . . . .	181
Adding the User Dictionary to the Desktop Application . . . . .	183
<b>Using the MapInfo User Dictionary Utility . . . . .</b>	<b>185</b>
<b>Configuring the User Dictionary . . . . .</b>	<b>187</b>
Configuring the DataManagerSettings.properties File . . . . .	188
Setting Dictionary Preferences Using MapMarker Java API . . . . .	189
<b>Preferring User Dictionary Matches . . . . .</b>	<b>191</b>
<b>JVM Settings . . . . .</b>	<b>194</b>
<b>Country and Coordinate System Settings . . . . .</b>	<b>194</b>
<b>MapMarker Java API Errors . . . . .</b>	<b>196</b>
Engine Errors . . . . .	196
Parser Errors . . . . .	196
Data Access Exceptions . . . . .	196
License Exceptions . . . . .	197
General Geocoding Process Exceptions . . . . .	197
Client Exceptions . . . . .	198
Server Exceptions . . . . .	198

---

<b>Geocoding Engine Errors</b> .....	<b>199</b>
<b>Adapter Errors</b> .....	<b>199</b>
<b>Index</b> .....	<b>203</b>

# Introduction

Welcome to MapMarker, the premier address matching product from Pitney Bowes MapInfo. 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
- ♦ **Getting Technical Support** .....14

# What Is Geocoding?

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

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

# What Is MapMarker?

MapMarker is a powerful geocoding tool produced by Pitney Bowes MapInfo. MapMarker can geocode large tables of U.S.-based street addresses in a single pass. This is a key 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.

The MapMarker Desktop Application, MapMarker server, and MapMarker developer tools have geocoding capabilities that allow you to add geocoding functionality to your own desktop or Web application.

## 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 address ZIP Code centroids or to street intersections.
- Identify result codes quickly tell how the address match was made and how precise the match is.
- Standardize addresses to meet CASS™ standards.

- Use DPV<sup>®</sup>.
- Geocode remote tables via ODBC.

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

## Beyond the Desktop

In addition to the MapMarker Desktop Application, MapMarker is available as a server product and developer tool. Using the Java API, developers can write client/server or standalone geocoding applications. The MapMarker server enables multiple simultaneous geocoding requests to be served from a single MapMarker engine.

The MapMarker developer tools include a Java API that enables you to write geocoding applications and an OCX for adding geocoding functionality to your own applications. If the OCX does not 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 Standard or MapMarker Plus

The MapMarker Desktop Edition is available in two versions: Standard or Plus. If you purchased MapMarker Standard, you receive the standard Address Dictionary. MapMarker Standard is released once 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<sup>®</sup> requirements for CASS, qualifying for address standardization and bulk mail discounts. MapMarker Plus with the Plus Address Dictionary is released quarterly.

## New in This Release

See the MapMarker 14.0 Release Notes for a summary of new features, address dictionary updates, and behavioral changes that you may notice with MapMarker 14.0. The MapMarker Release Notes are located at <http://reference.mapinfo.com/>.

The Release Notes also describe fixes to customer-reported issues and known issues that you must be aware of.

The data vintage and version information has been added to the About MapMarker dialog (**Help > About MapMarker**). For complete data vintage compatibility information on related products, see the [Data Vintage Chart](#) on the Pitney Bowes MapInfo web site. The Help About dialog also reports the DPV expiration date.

## ParcelPrecision<sup>™</sup> User Dictionary

MapMarker Plus 14 is synchronized with the ParcelPrecision data product. ParcelPrecision is a set of custom user dictionaries containing center points of actual parcels as well as the locations of those centroids as positioned on the street associated with each parcel.

This data, when used with MapMarker 14.0 address point interpolation capabilities, or the Envinsa Location Utility Service, has the premier address point geocoding precision that is vital for insurance, public sector, homeland security, and other sectors.

**Note** When using CASS geocoding, user dictionaries (including ParcelPrecision) are ignored.

### 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 Plus or Envinsa 4.0, or later as a geocoding service from within MapInfo Professional 8.5, or later. This direct connection enables users to take advantage of MapMarker geocoding capabilities within the powerful MapInfo Professional mapping and analysis environment.

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

## MapMarker Documentation Set

The documentation set for MapMarker Standard and MapMarker Plus includes PDF and online resources to help you make the most of the product. The set includes:

- MapMarker User Guide in PDF format. The MapMarker Plus User Guide is also provided in hard copy.
- Online Help for the MapMarker Desktop Application.
- MapMarker Developer Guide in PDF format (if you purchased the MapMarker Developer Edition).
- Release Notes that provide updated information on new features, behavioral changes in the software, fixes for customer-reported issues, and known issues. MapMarker Release Notes are posted to <http://reference.mapinfo.com/>.

### MapMarker 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. Coverage includes:

- Geocoding in MapMarker
- Configuring match settings for optimum geocoding results
- Explaining result codes
- Geocoding remote tables
- Creating a customized user dictionary

The User Guide also contains a chapter on more specialized features, including CASS™ geocoding and DPV®; finding airports, highway exits, single address geocoding, batch geocoding, and table attribution.

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

This guide also includes appendixes that provide reference information for MapMarker.

- MapMarker Preferences—Contains 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.
- U.S. ZIP Code™ Ranges—A list of the ZIP Code ranges in the United States.
- 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 dialogs in MapMarker.

## MapMarker Release Notes

The MapMarker Release Notes are posted to <http://reference.mapinfo.com/>, and contain a brief summary of the following:

- New features
- Behavioral changes
- Bug fixes
- Known issues

## Publications on the Web

The MapMarker documentation and Release Notes are available for download on the Pitney Bowes Mapinfo web site (<http://www.mapinfo.com>). From the home page Support and Training tab, click **Documentation** 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 Application. Online help is instantly available while you are running MapMarker or the installation wizard. To access help, either select the Help menu, press the F1 key, or click **Help** for help about a dialog.

## MapMarker Streets

Once you have geocoded your table and are ready to display it in MapInfo Professional, you may 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 MapMarker Plus geocoded data.

MapMarker Streets is provided on separate media with your MapMarker product. See **Installing MapMarker in Chapter 2 on page 17** for instructions on installing MapMarker Streets.

## Getting Technical Support

Pitney Bowes MapInfo offers a free support period on all new software purchases and upgrades. Once the free period ends, Pitney Bowes MapInfo offers a broad selection of extended support services for individual, business, and corporate users.

Technical Support is available 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 know what to expect about the handling and resolution of your particular issue.

Pitney Bowes MapInfo has full technical support for MapMarker for versions 11.0 and later. 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 company holidays. Closed between 10:30am - 11:30 am on Mondays for training.

### *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 company 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 company 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 company holidays.

To use Technical Support, you must register your product. This can be done very easily during installation. To receive more information on 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 technical support or renew your current contract, contact Customer Service at 1-800-552-2511, and follow the voice mail instructions, or send an e-mail to [custserv@mapinfo.com](mailto: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**

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 recurring 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 has 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 is issued before the end of the business day. A Technical Support representative provides 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 may ask you to provide sample data to duplicate your scenario. In the case of our developer tools, 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 Pitney Bowes MapInfo FTP site

For information regarding our FTP site, 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 recognized as a bug in the software, the Technical Support representative logs the issue in our bug base and provides you with an incident number that can be used to track the bug.

## Customer Service (Non-Technical Support)

To resolve questions about the accounting of your 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** .....18
- ♦ **Operating Requirements and Performance Recommendations** .19
- ♦ **Installing MapMarker** .....21
- ♦ **Modifying the MapMarker Installation** .....27
- ♦ **Upgrading the MapMarker Installation** .....28
- ♦ **Uninstalling MapMarker** .....29
- ♦ **Client Workstation Install** .....30
- ♦ **Starting MapMarker** .....31

# Installation Overview

MapMarker is available as a Desktop product and as a Server product. The features available for installation are determined by the edition that was purchased. The features available during installation are as follows:

**MapMarker Desktop Product** The Desktop product includes the Desktop Application which includes the graphical user interface for geocoding addresses and the MapMarker Address Dictionary. You can purchase data for all 50 states (plus D.C. and Puerto Rico), regional state bundles, or individual state data.

The Desktop Application runs on Microsoft Windows only.

**MapMarker Server Product** The Server product enables multiple simultaneous geocoding requests to be served from a single MapMarker engine. It also includes the tools necessary for developing MapMarker solutions using the Java API or XML API. These tools include the MapMarker SDK, native code libraries, and examples. A preconfigured Apache Tomcat servlet is provided to run MapMarker as a geocoding server.

The Server product also includes one license for the MapMarker Desktop Application.

MapMarker is delivered on one DVD, which includes both the software and the data. An installer is provided to install and configure the product in one easy procedure.

The MapMarker installer program can be used for the following purposes:

- New installation. See [Installing MapMarker on page 21](#).
- Modifying existing installation to add or remove features. For example, you could add new data or add a feature to an existing installation. See [Modifying the MapMarker Installation on page 27](#).
- Upgrading existing installation. For example, when a point release version of MapMarker is available, you could upgrade from version 13.1 to 13.2 with the same set of features that are already installed. See [Upgrading the MapMarker Installation on page 28](#)
- Uninstalling. See [Uninstalling MapMarker on page 29](#).

## Shared Install

The MapMarker data can be shared among a group of users who wish to run the MapMarker Desktop Application. To do so, the system administrator does a shared installation of MapMarker. This must include the Desktop Application feature and 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.

After the system administrator has made the shared program and data available across the network, client users can browse to the program location and run the Desktop Application installer (ClientSetup.exe) to install the MapMarker Desktop Application on their workstations. Client users can then run the MapMarker Desktop Application and access the shared Address Dictionary.

For instructions on installing the client Desktop Application for use with a shared installation, see [Client Workstation Install on page 30](#).

## About the Address Dictionary

The MapMarker Address Dictionary is installed during the installation of the MapMarker software. You can install data for individual states, the entire United States, or ZIP + 4<sup>®</sup> centroids. The Address Dictionary must be stored on a local hard drive or a network drive. By default, the data resides in the data directory under the MapMarker program directory.

You may choose to install all or only a portion of the data set that you purchased. To install only a portion of the purchased data set, uncheck states that you do not want to install when you reach the Choose Install Set dialog.

## Operating Requirements and Performance Recommendations

Note the following recommendations to maximize the performance of your MapMarker installation.

[Operating Requirements on page 19](#)

[Optimizing Performance on page 21](#)

### Operating Requirements

This section describes the basic software and hardware requirements for MapMarker. See your version-specific Release Notes for more detailed descriptions and information on supported software and hardware.

These guidelines may change for later versions of MapMarker Plus 14.x. See the product Release Notes for updated operating requirements for your specific version of MapMarker.

#### Hardware and Memory Requirements

The minimum system requirements for MapMarker on a full installation of the Desktop Application and MapMarker SDK are:

- 800 Mhz Pentium<sup>®</sup> processor or equivalent
- 512 MB RAM
- 5 GB available disk space (including 4GB for entire U.S. Address Dictionary)

For optimal performance of a full installation of both the Desktop Application and MapMarker SDK we recommend:

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

### Supported Operating Systems

MapMarker runs on the following operating systems. You must see your version-specific MapMarker Release Notes for detailed information on versions, platforms, and kernels supported by your MapMarker product.:

- Microsoft Windows
- Solaris
- HP-UX
- Red Hat Linux
- SUSE Linux

See the product Release Notes for detailed information on supported operating systems for your specific version of MapMarker.

### JVM Requirements

MapMarker requires a Java Virtual Machine (JVM) to run. You must see your version-specific MapMarker Release Notes for detailed JVM support information for your MapMarker product. MapMarker Plus 14 ships with the following:

- Sun JRE v. 1.5.0\_09 (for Microsoft Windows and Sun Solaris)
- HP JRE 1.5.0\_04 (for HP-UX)
- Sun JRE 1.5.0\_11 (for Linux)

See the product Release Notes for detailed JVM support information for your specific version of MapMarker.

### Database Requirements

MapMarker Plus 14 supports the following ODBC databases and drivers. You must see your version-specific MapMarker Release Notes for detailed database support information (including Microsoft Access, Oracle, and SQL Server).

- SQL Server
- Microsoft Access 2007
- Oracle

See the product Release Notes for updated database support information for your specific version of MapMarker.

### Web Server Requirements (Server Product Only)

MapMarker Plus 14 ships with Apache Tomcat 5.5 (installed if the Web Application feature is selected during the Developer Product installation).

The following web application servers are also supported:

- Oracle 10G Application Server
- BEA WebLogic
- WebSphere

See the product Release Notes for detailed web server support information for your specific version of MapMarker.

## Optimizing Performance

Consider these guidelines for optimizing MapMarker performance:

- Use the fastest processor available to you. We recommend 2.5 Ghz dual processor or better.
- Have enough memory so that the operating system can allocate some memory to your disk cache. We recommend 1 GB RAM or better.
- Have at least 10 GB available disk space.
- Sort your table by ZIP Code™ before geocoding.
- Choose exact match criteria for all (house number, street name, city name, ZIP Code).
- Do not create points automatically.
- Remove indexes on any table output columns before geocoding.
- Store the Address Dictionary on a local hard drive.

## Installing MapMarker

This section describes the typical MapMarker installation procedures for Microsoft Windows users and for UNIX/Linux users. The graphical installer interface guides you through the procedure. You can also install silently, without user interaction.

[Install Using the Graphical Interface on page 21](#)

[Installing Silently on page 26](#)

### Install Using the Graphical Interface

Follow these steps for typical installation using the graphical installer program:

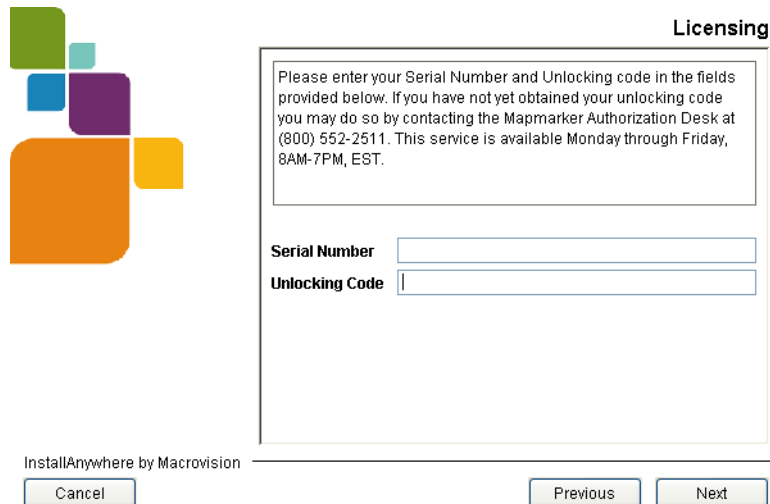
1. Begin the installation procedure as follows, depending on whether you are installing on Windows or on UNIX/Linux:
  - On Microsoft Windows, place the MapMarker DVD into your DVD-ROM drive. The installation starts automatically. Click **Install Product** in the list of options. In the Install Product dialog, click **Install MapMarker Plus USA**. Proceed to [step 2](#).  
If the installation does not automatically run, from the Windows 2000/2003/XP Start menu, select **Run**. From the Run dialog, type `D:\SETUP.EXE` in the Open command box (where D is the drive letter of your DVD-ROM). On Microsoft VISTA, from the Windows VISTA Start menu, type `D:\SETUP.EXE` in the search box provided, where D is the drive letter of your DVD-ROM.
  - On Solaris, Linux, or HP-UX, run the shell script from the root of the DVD-ROM using the command:
 

```
sh startinstall.sh
```

 Proceed to [step 2](#).  
To bypass the startinstall.sh script, go to the /instdata folder of the DVD-ROM, and run the executable appropriate for your operating system. For example, on a Linux operating system:
 

```
sh linux/install.bin
```
2. In the Introduction dialog, click **Next** to display the What's New dialog. This has a brief summary of new features for MapMarker Plus 14 and has data vintage information.
3. Click **Next** to display the License Agreement dialog.

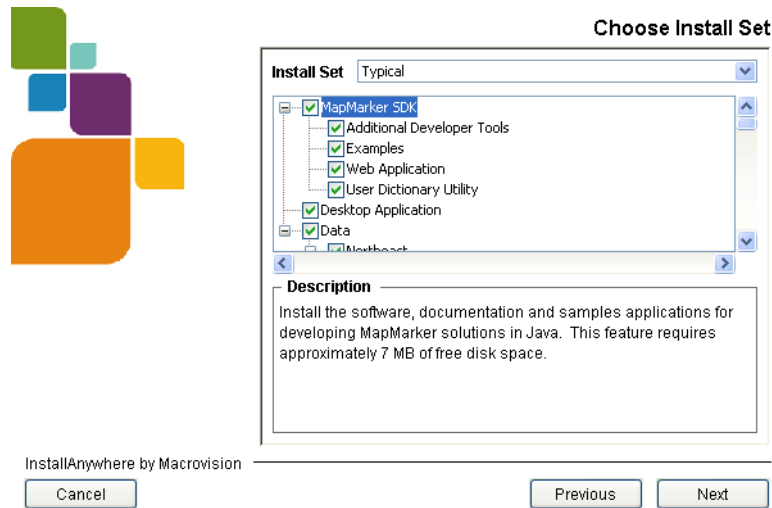
4. After reading the License Agreement, click **I accept the terms of the License Agreement** and then click **Next** to display the Licensing dialog.
5. In the Licensing dialog, type your MapMarker serial number and unlocking code. The serial number is located on a sheet of stickers in the MapMarker package. The unlocking code enables the installer to create a license file for your system that gives you access to the MapMarker features and 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 has your unlocking code. Have your product serial number ready when you call.



After entering the serial number and unlocking code, click **Next**.

If you entered valid serial number and unlocking codes, you see the Choose Install Set dialog. If you did not enter the correct codes, you receive an error message and have the opportunity to reenter the serial number and unlocking code.

6. In the Choose Install Set dialog, the default, Typical Install Set, shows all the software and all the data you purchased. To see a description of each feature, click on the feature to highlight it. A description of the selected feature appears in the Description box.

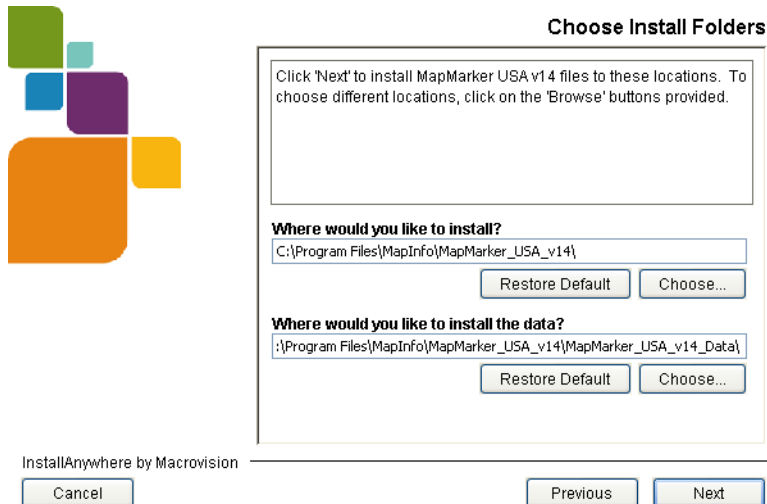


Check the boxes for the features that you want to install and clear the check boxes of the features that you do not want to install. Product documentation is always installed.

**Note** The MapMarker SDK selections are available only for the Developer Edition.

After choosing the features to install, click **Next** to display the Choose Install Folders dialog

- In the Choose Install Folder dialog, select the locations to install MapMarker software and data.



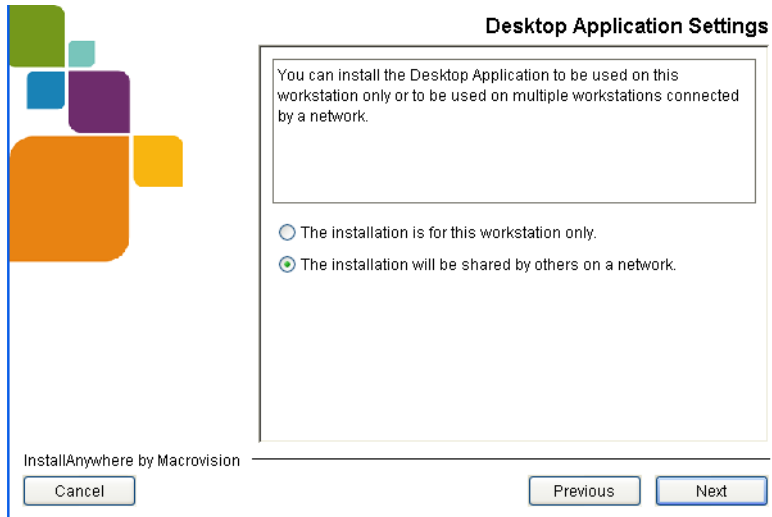
You can use the default locations, or click **Choose** to display the Browse for Folder dialog. From the Browse folder you can navigate to the desired location. Click **OK** in the Browse for Folder dialog when you are finished.

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.

When you have selected the software and data install folders, click **Next**.

8. If you checked the Desktop Application feature in the Choose Install Set dialog (in [step 6](#)), the Desktop Application Settings dialog appears. If you are not installing the Desktop Application, skip to [step 11](#).

In the dialog, choose whether the installation is for this workstation only or whether the installation is shared by others on the network.



Select **The installation is for this workstation only** if you want a local installation of MapMarker software and data. This is not shared with other users.

Select **The installation will be shared** if this workstation of MapMarker serves as a host that provides a group of users on the same network a common location for the software and data. Users who wish to access the shared software and data that resides on the host workstation must do a separate client installation. See [Client Workstation Install on page 30](#) for more information.

After choosing a workstation only or shared installation, click **Next** to continue.

9. If you chose that the installation would be Shared ([step 8](#)), the dialog asks if you want to install the MapMarker Desktop Application on this workstation (the host workstation). If you chose that the installation would be for this workstation only, proceed to [step 11](#).

If you plan to use the MapMarker application on this workstation (the one on which you are doing the Shared installation), check **Install the Desktop Application on this workstation**. If you are installing the software only for other users to access, do not check this box.

Click **Next** to continue.

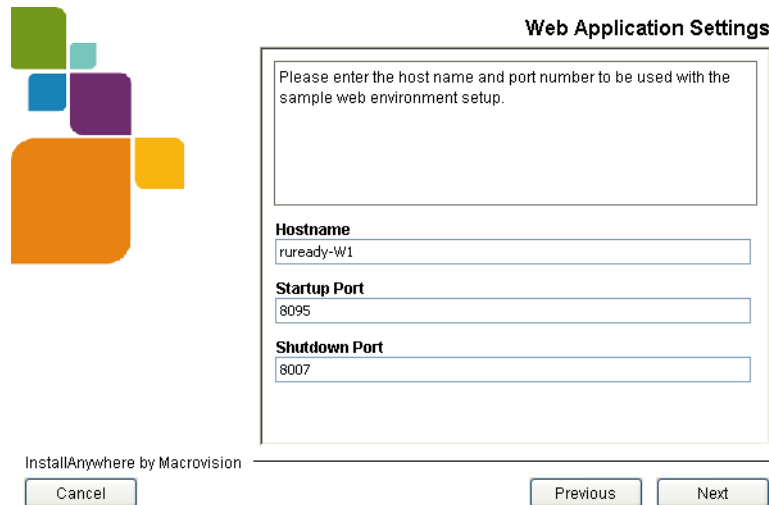
10. The Desktop Application Settings dialog now shows the path to the shared data folder. Confirm or correct the path to the Shared Data Folder. Universal Naming Convention (UNC) paths are preferred (for example: \\ruready-W1\MapMarker\_USA\_v14\Data). This ensures that users running MapMarker on other machines are able to access the shared data. If the network share name or shared folder does not already exist, it must be created before you can use the MapMarker application.

**Note** The install program does not ensure that the data folder is shared to others on the network or is visible on the network. The person doing the installation is responsible for entering a valid path to the data folder and making this data folder accessible.

After you have confirmed the shared data folder, click **Next** to continue.

11. If you are installing the MapMarker SDK and you chose to install the Web Application feature (from the Choose Install Set dialog, [step 6](#)), the Web Application Settings dialog appears. If you did not select this feature, proceed to [step 12](#).

This dialog gives you the opportunity to confirm or change host name, startup port, and shutdown port. Port number 8095 is the default Startup Port number. Port 8007 is the default Shutdown Port number.



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

12. In the Choose Java Virtual machine dialog, do one of the following:
- Select **Install a Java VM specifically for this application** to install the Java virtual machine that comes with MapMarker.
  - Select **Choose a Java VM already installed on this system** to select a Java virtual machine that you already have installed on the computer. Then select **Search for Others** to search your computer for an installed JVM. After the search is completed, the dialog is populated with a list of the JVMs that were found on your workstation. Select the JVM that you want to use.
- Note** See your MapMarker version-specific Release Notes for complete information on operating requirements and recommendations, including information on the supported and recommended JVMs.

After selecting the JVM, Click **Next** to display the Choose Shortcut Folder dialog. If you select a JVM that is already installed on your system, and that JVM is invalid or cannot be found, you receive an error message.

13. In the Choose Shortcut Folder dialog, choose where you want to create the product icons. You can create a new Program Group, place the icons in an existing Program Group, or make other choices. After making your selection, click **Next** to continue.
14. In the Pre-Installation Summary dialog, review the disk space requirements and your install selections. To edit any selections, click **Previous** to go back to the earlier dialogs.

15. When you are satisfied with your installation choices, click **Install**. A progress dialog reports progress 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. New files are added as the process continues.
16. When the installation is finished, the Install Complete message appears. Click **Done** to exit the installer.

## Installing Silently

You can install MapMarker Plus 14 without using a graphical interface and without user interaction. This allows you to install MapMarker in environments that do not provide a graphical user interface.

To do a silent install, use the properties file located in the root folder on the DVD. This file defines the values for the installer to use when installing in silent mode. The file is named:

silentinstall.properties

You must edit the file to suit your particular needs. This file includes extensive comments to assist you in this process. The serial number and unlock code variables must be populated with valid values. By default, the file is initially configured to install the engine, web application, samples, additional tools, and licensed data. After editing this file, you can then launch the silent install and pass this properties file to the installer program.

To edit the silentinstall.properties file:

1. Copy the silentinstall.properties template file to a location on the target machine 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 and Unlocking Code
  - Install location of the application files and data
  - Java Virtual Machine
  - Shortcuts/links location
  - Features to install
  - Web Application configuration
  - Desktop Application configuration
3. To launch the silent install, use the following command line arguments:

On Unix/Linux (where *media/CDROM* is the location of the installation media and */tmp* is the location of the silentinstall.properties file).

```
/media/CDROM/startinstall.sh -f /tmp/silentinstall.properties
```

On Windows operating systems (where D: is the drive location of the installation media and C:\tmp is the location of the silentinstall.properties file).

```
start /w D:\instdata\install.exe -f C:\tmp\silentinstall.properties
```

**Note** You must use an absolute path for the properties file.

## Modifying the MapMarker Installation

After you have installed MapMarker, you can run the Install program to modify the MapMarker installation. For example, you may want to add data for states that you originally purchased but did not previously install. Or you may want to remove features that you had previously installed. These changes are characterized as modifications.

**Note** To modify your installation, you must use the same installer program that was originally used to install MapMarker. If you use a different installer program (for a different version of MapMarker) you will not be able to modify the installation.

A modification is distinct from an upgrade. An upgrade involves installing a point release (such as 14.2) on top of the major (14.0) or previous point release (14.1). A modification adds features to or removes features from the currently installed version.

**Note** Some things cannot be modified, such as the paths for software and data.

Since MapMarker is already installed on your system, the install program opens to the Introduction dialog. This bypasses the Licensing dialog, since you have already accepted the License Agreement. You can then proceed to modify your installation.

**Modify Using the Graphical Interface on page 27**

**Modify Silently on page 28**

### Modify Using the Graphical Interface

To modify your MapMarker Plus 14 installation, do the following:

1. Start the installer as described in **step 1 of Install Using the Graphical Interface on page 21**.
2. At the Introduction dialog, click **Next**.
3. Enter or confirm your serial number and unlocking code, then click **Next**. For a modification, this information is already populated in the dialog. If you purchased new features (such as additional data), you have a new unlocking code and must enter this in the Licensing dialog.
4. The Install Set dialog appears and shows a checked box for every MapMarker feature that is installed. This is where you can install licensed features that you previously chose not to install, or uninstall selected features. Do the following:
  - a. Uncheck any feature that you want to remove. For example, you can uncheck and remove Northeast data (or data for selected states) that you previously installed. Click **Next**.
  - b. Check features that you want to install. You see only features that are licensed to you, based on your unlocking code. Click **Next**.
5. You see the Desktop Application Settings or Web Applications Settings dialogs if you chose to install the Desktop Application or Web Application. See **step 8** (Desktop Application) and **step 11** (Web Application) from the topic **Install Using the Graphical Interface**.
6. The Pre-installation Summary dialog shows a list of product features to be installed. If you chose to install new features, they are listed. If you chose to uninstall any features, those features are absent from the list of product features.
7. After verifying the product features, click **Install**.
8. The Install Complete dialog confirms your actions. Click **Done**.

### Modify Silently

You can modify MapMarker Plus 14 without using a graphical interface and without user interaction. To do a silent modify, edit the `silentinstall.properties` file to suit your needs. See [Installing Silently on page 26](#) for information on the location and use of the properties file.

The `FEATURE_LIST` variable of the `silentinstall.properties` file must include **all** the MapMarker features that you want to install. For example, if you already had NY, TX, and CA data installed and wanted to add PA, FL, and AZ, make sure that the `FEATURE_LIST` indicates all of those states – not just the states you are adding.

You cannot use the silent modify procedure to change any of the following settings:

- Program paths
- JVM
- Shortcuts and links

You cannot use the silent modify procedure to change the configuration of a previously installed Web Application (for Developer Edition). Similarly, you cannot change the shared/unshared configuration of the previously installed Desktop Application. If you want to make any of these changes in your installation, you must uninstall then reinstall MapMarker.

## Upgrading the MapMarker Installation

Major releases (such as 14.0) can be upgraded when point releases (such as 14.1) become available. Similarly, one point release can be upgraded to the next point release. When doing an upgrade, the installer updates all of the currently installed MapMarker features to the new version. All settings and configurations remain unchanged.

If you want to add or remove features or change settings, you must first upgrade and then run the installer again to modify your installation (see [Modifying the MapMarker Installation](#)). Alternatively, you may uninstall and then re-install MapMarker if you have changes that cannot be done using the Modify option.

**Note** You **cannot** upgrade from one major release to another. For example, you cannot upgrade from MapMarker 13.x to 14.0. You must do a new installation of MapMarker 14.0. You can then upgrade from version 14.0 to the 14.x point release.

You can have different major releases on the same workstation (such as MapMarker Plus USA Version 13 and Version 14), but you cannot have different point releases installed on the same workstation (For example, version 14.0 and version 14.1 cannot be on the same workstation.) Also, the same versions of MapMarker Standard and MapMarker Plus cannot be installed on the same workstation. We recommend that you contact Technical Support if you want to use more than one version of MapMarker.

[Upgrade Using the Graphical Interface on page 29](#)

[Upgrade Silently on page 29](#)

## Upgrade Using the Graphical Interface

To upgrade from MapMarker 13.x to a point release (such as to 14.0), do the following:

1. Start the installer as described in [step 1 of Install Using the Graphical Interface on page 21](#).
2. In the Introduction dialog, click **Next** to display the What's New dialog.
3. In the What's New dialog, click **Next** to display the License Agreement dialog.
4. After reading the License Agreement, click **I accept the terms of the License Agreement** and then click **Next** to display the Licensing dialog.

Because this is an upgrade, you will not be prompted to select installation features, folders, or application settings. Nor will you be asked for serial number and unlocking codes. All of this information is gathered from the original installation, and cannot be changed during an upgrade. If you want to change the installation feature set, you must modify (rather than upgrade) the installation. See [Modifying the MapMarker Installation on page 27](#).

5. In the Pre-Installation Summary dialog, click **Install** to begin the upgrade.
6. When the Upgrade Complete dialog appears, click **Done**.

## Upgrade Silently

You can upgrade MapMarker Plus 14 to a point release without using a graphical interface and without user interaction. Start the silent installer as described in [Installing Silently on page 26](#). Any changes to the installation reflected in the properties file are ignored by the silent upgrade.

## Uninstalling MapMarker

When uninstalling MapMarker, you have two options: a complete uninstallation, or a removal of certain features only. A complete uninstall removes the MapMarker program directories, groups, and icons. Files and folders created after the installation are not removed or affected in any way.

You can uninstall specific features (rather than a complete uninstall) if you use the graphical interface uninstall. The silent uninstall always does a complete uninstall.

Uninstall must be done using the same mode (graphical interface or silent) as the previous installation, modify, or upgrade. So if your previous install, modify, or upgrade was done using the graphical interface, then the uninstall is run using the graphical interface.

[Uninstall Using the Graphical Interface on page 30](#)

[Uninstall Silently on page 30](#)

### Uninstall Using the Graphical Interface

If MapMarker was installed as a shared installation, the uninstaller removes the MapMarker Desktop Application if it was previously installed on that workstation. Each user who accesses the shared application must use Add/Remove Programs on their own workstation to remove the MapMarker Desktop Application.

1. Begin the uninstall procedure as follows, depending on whether you are uninstalling on Windows or on UNIX/Linux:

- On Microsoft Windows, from the Control Panel **Add or Remove Programs** utility select MapMarker USA v14 and click **Change/Remove**. Proceed to **step 2**.
- On Solaris, Linux, or HP-UX Run the shell script from the root of the DVD-ROM using the command:

```
sh /opt/MMv14/Uninstall_MapMarker_USA_v14/Uninstall_MapMarker_USA_v14
where /opt/MMv14 is the directory into which MapMarker Plus 14 was installed.
```

2. In the Uninstall MapMarker USA dialog, click **Next**.
3. Select **Complete Uninstall** or **Uninstall Specific Features**, then click **Next**.
4. If you selected **Complete Uninstall** in **step 3**, the uninstall program runs immediately and MapMarker Plus 14 is completely uninstalled.  
If you selected **Uninstall Selected Features** in **step 3**, uncheck the product features that you want to uninstall, then click **Uninstall** to uninstall those features only.
5. When the uninstall completes, click **Done**.

### Uninstall Silently

If your previous installation, modify, or upgrade of MapMarker was done silently, the uninstall runs silently as well. Launch the uninstall by running the command appropriate for your operating system shown in **step 1** of **Uninstall Using the Graphical Interface**. Remember, a silent uninstall always does a complete uninstall of MapMarker.

## Client Workstation Install

If the system administrator has done a Shared installation of MapMarker and shared the program folder and data folder across the network, users on the network can run client installations on their local workstations. Client users can then run the MapMarker Desktop Application and access the shared Address Dictionary.

For instructions on doing a Shared installation, see **Shared Install on page 18**.

To set up a client workstation to access MapMarker from a shared installation, follow these steps:

1. Browse to the location of the shared installation of MapMarker on the network server. (You may need to check with the System Administrator for the location.)
2. From the shared machine, run the setup program (ClientSetup.exe) in the desktop folder of the MapMarker program directory. The Welcome dialog appears.
3. In the Welcome dialog, click **Next** to display the Destination Folder dialog.

4. You can click **Next** to accept the default destination folder or click **Change** to display the Change Current Destination Folder dialog and navigate to another location. Click **OK** in the Change Current Destination Folder dialog when you have specified the new location, then click **Next** to display the Ready to Install the Program dialog.
5. At the Ready to Install the Program dialog, click **Install**.
6. When the installation wizard completes, click **Finish**.

All the components necessary to run the MapMarker Desktop Application using the shared Address Dictionary are installed. This includes MDAC and the Microsoft Jet engine with their desktop ODBC drivers.

To uninstall the client application, from the Control Panel select the **Add or Remove Programs** utility. Select MapMarker 14 Desktop and click **Change/Remove**.

## Starting MapMarker

After successfully installing MapMarker Plus 14, you can start the Desktop Application, if you installed that feature on your workstation, or start the MapMarker Server if you installed the Web Application.

[Starting the Desktop Application on page 31](#)

[Starting the MapMarker Server on page 31](#)

### Starting the Desktop Application

If you installed the Desktop Application feature of MapMarker, you can start the MapMarker Desktop as follows:

- From the Start Menu or the program group where you installed the MapMarker icons, click the icon named **MapMarker Desktop**. The Desktop Application starts.  
If you chose not to install icons, open the directory where you installed MapMarker, and in the subdirectory named `desktop` double-click on the file `mapmarkr.exe`. The Desktop Application starts.

### Starting the MapMarker Server

If you installed MapMarker Plus 14 Web Application feature (available with the Developer Edition), you can start the MapMarker server as follows, depending on whether you are installing on Windows or on UNIX/Linux:

- On Microsoft Windows, from the Start Menu or the program group where you installed the MapMarker icons, click the icon named `Start MapMarker USA v14 Server`. The MapMarker Server starts.  
If you chose not to install icons, open the directory where you installed MapMarker, and in the subdirectory named `sdk\tomcat\bin` double-click on the file `startup.bat`. The MapMarker Server starts.
- On Solaris, Linux, or HP-UX, from the directory where you installed the MapMarker icons, run:

## Starting MapMarker

---

```
sh Start_MapMarker_USA_v14_Server
```

If you chose not to install icons, run the following command from the installation directory:

```
sh sdk/tomcat/bin/startup.sh
```

**Note** On all operating systems, Administrator privilege is required to start the MapMarker Server.

# 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** .....34
- ♦ **Geocoding Terms** .....36
- ♦ **Defining the Geocoding Model** .....39
- ♦ **Geocoding Application Design Overview** .....40

## Choosing a Development Tool

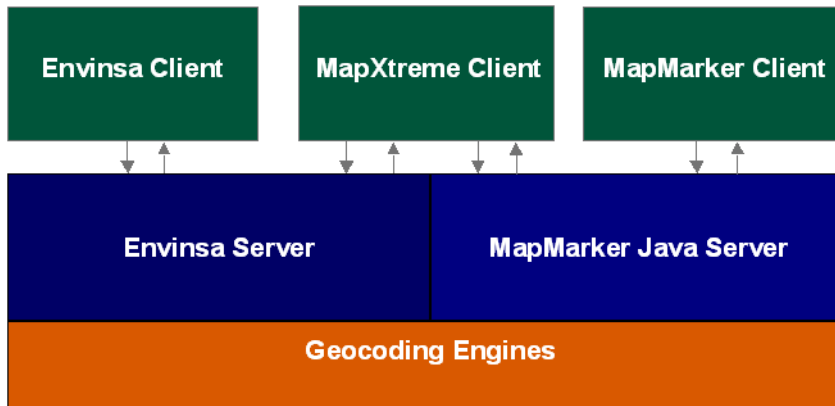
When you create a geocoding application, your choice of development tool will have a direct effect on the design and deployment of your application. As you are evaluating tools, some questions to consider include:

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

Pitney Bowes 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.

## Geocoding Interoperability

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



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

The following table summarizes the geocoding solutions available.

Product	Web/Desktop*	OS	Programming Language	Capabilities
MapMarker Plus 14	Both	Windows, Solaris, HP-UNIX, Linux	Java	Geocoding
MapXtreme	Both (IIS)	Windows	.NET	Geocoding, Mapping, Routing
Envinsa	Both (App Server)	Windows, Solaris, Linux	Java, XML, .NET, and Web Services	Geocoding, Mapping, Routing, and more

The following sections provide a summary of some of the features and design capabilities of geocoding offerings.

### MapMarker Plus 14.0

- Desktop and Developer Editions. The Desktop Edition includes the Desktop Application graphical user interface and Address Dictionary. You can purchase data for all 50 states (D.C. and Puerto Rico), regional state bundles, or individual state data. The Developer Edition includes the Desktop Application, the MapMarker SDK, Java API, a JNI Adapter to run the OLE Automation API and Data Dictionary API, and examples. A preconfigured Apache Tomcat servlet is provided to run MapMarker as a geocoding server. Both the Desktop and Developer Editions include complete documentation. Javadocs is included with the Developer Edition to document the MapMarker Java 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.
- 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

## Geocoding Terms

---

- 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).

## Geocoding Terms

MapMarker products use a common set of geocoding terms.

### Geocoding Terminology

Term	Definition
address	Information that identifies the location of a site, for example, a home or business. An address is either a street location or intersecting streets.
address dictionary	The search dictionary used for matching addresses during geocoding.
address interpolation	A mathematical means of estimating an address location based on a street segment and the address ranges associated with that segment. 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.
candidate	An address record that is a potential match to the input address information.  A candidate is returned by the geocoding process. See also <a href="#">geocoding</a> .
CASS™	Coding Accuracy Support System. This is a process by which mailing addresses are standardized to meet U.S.Postal Service® requirements for bulk mailing discounts. When CASS mode is enabled, MapMarker Plus will perform this address standardization under strict matching conditions set by the USPS® when it geocodes your records. Geocoding in CASS mode matches records to the exact house number, street name, and ZIP Code™. Other geocoding preferences are overridden.
centroid	A point at the calculated center of a polygon, often used to attach attribute information to an area. The centroid is used for labeling and geocoding, for example.  For an irregularly shaped area, region, or polygon, the centroid is derived mathematically and is weighted to locate the point at the approximate "center of gravity."

Geocoding Terminology (*continued*)

Term	Definition
close match	Status that indicates whether a candidate's ranking is considered close enough to the input address.
CMRA	Commercial Mail Receiving Agencies
DPV <sup>®</sup>	Delivery Point Validation. DPV is technology available from the U.S. Postal Service <sup>®</sup> . Used in conjunction with CASS <sup>™</sup> mode geocoding, DPV enables you to verify whether an address is a deliverable USPS <sup>®</sup> address.
geocoding	<p>The process by which the X and Y coordinates of a location are determined by its address.</p> <p>In MapMarker, the X and Y coordinates are longitude and latitude.</p>
geographic geocoding	Candidate coordinates are based on state, county, or city centroids. This not very precise, but may be suitable for certain applications.
LACS <sup>Link®</sup>	<p>Locatable Address Conversion System. LACS<sup>Link</sup> allows business mailers to electronically update their rural-style addresses with locatable street-style addresses resulting from 911 emergency response address conversions.</p> <p>For example, an address like RR 1 Box 32 might be converted to a street-style address like 141 Morrison Ave.</p>
match strategy	<p>An approach to geocoding a table that achieves the desired geocoding results. For example, if geocoding hit rate is more important than accuracy, then geocoding criteria would be relaxed to ensure that more records in the table are geocoded.</p> <p>Or if you required greater accuracy, you could use stricter geocoding criteria, such as an exact match on street name and house number.</p>
must match	Must Match options are used for deciding whether returned candidates are Close Matches. A Must Match option does not eliminate non-Close matches.
PMB	Private Mailbox

Geocoding Terminology (*continued*)

Term	Definition
postal geocoding	Candidate coordinates are based on postal codes. This is faster but not as accurate as street-level geocoding. Also, postal geocoding of rural areas is generally less accurate than that of urban areas. For P.O. Box™ and rural route addresses, MapMarker automatically geocodes to a ZIP Code™ centroid, as required by CASS standards.
postcode	A unique identifier for postal mailing zones.  For the U.S.A. this is the ZIP Code system. See <a href="#">ZIP Code™ on page 39</a> .
post-directional	The letters following a street name that give a direction to the address, for example 18th Street N. This shows that the location is on the north side of 18th street.  Not every address has a post-directional component.
post-type	Street type that follows the street name. For example, <b>Main St</b> .  Not every address has a post-type component.
pre-directional	The letters preceding a street name that give a direction to the address, for example W 18th Street. This shows that the location is on the west side of 18th street.  Not every address has a pre-directional component.
pre-type	Street type that precedes the street name. For example, <b>Avenue B</b> .  Not every address has a pre-type component.
result code	An alphanumeric code that describes the type and accuracy of the geocoding match.  These codes are returned when geocoding and allow you to identify the success of the geocoding request. These are described in more detail in <a href="#">Result Codes in Chapter 8 on page 109</a> .
street level geocoding	Candidate coordinates are interpolated along a street segment. This is a highly accurate level of geocoding.
street name	The name of the street. For example, <b>Main St</b> .

**Geocoding Terminology (continued)**

Term	Definition
user dictionary	A customized search dictionary of addresses and coordinates that can be used in place of, or in addition to, the application's address dictionary.
ZIP Code™	The system of 5-digit codes that identifies the individual post office or metropolitan area delivery station associated with an address.

## 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 USA 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 geographic centroid (state, county, or city) 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 44](#).

### 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<sup>®</sup> centroid or ZIP Code centroid is the best way to accomplish the user's task.

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

### Executing the Application

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

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

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

## Geocoding Application Design Overview

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

- Select a geocode type
- Build input address
- Set geocoding preferences
- Send a request
- Verify response
- Use candidate information

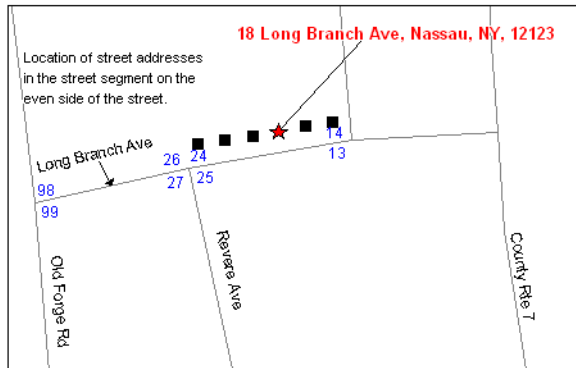
When you design your geocoding application, focus on each of the steps to determine the overall flow of the geocoding application.

## Types of Geocoding

MapMarker can perform street, postal, and geographic geocoding. The type of geocoding you want your application to perform depends on the needs of your users and how much positional accuracy they require.

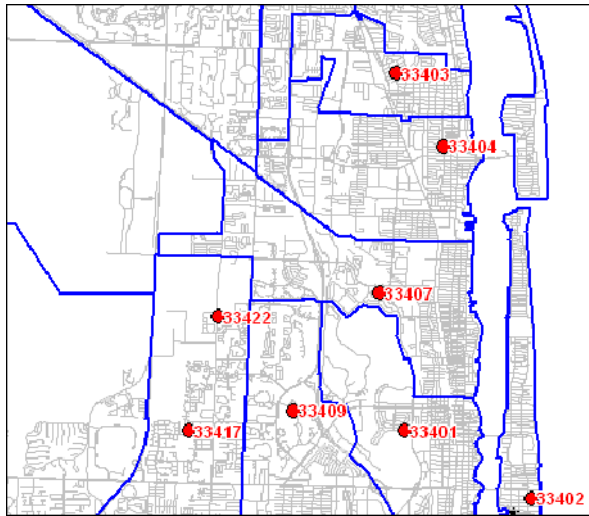
### Street Geocoding

In street geocoding, candidate coordinates are interpolated along a street segment. Geocoding to street level is highly accurate, but not as accurate as parcel-level accuracy, which can be achieved with 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

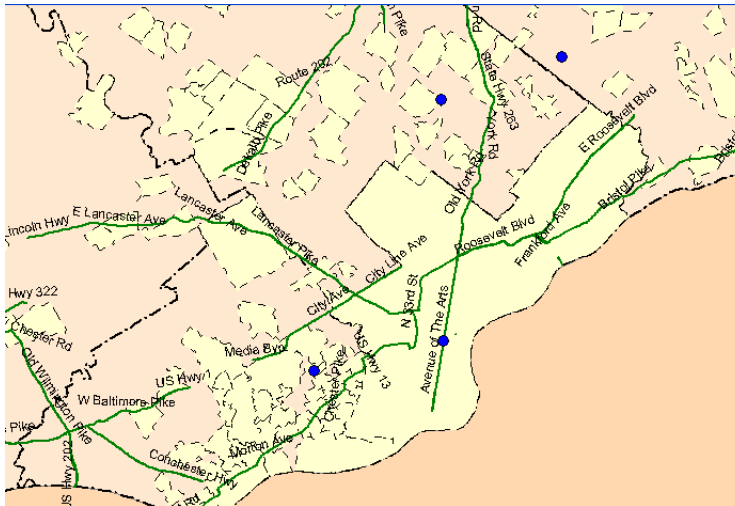
Postal level geocoding is faster but not as geographically accurate. Urban and rural accuracy can produce different results. This image shows the location of postal centroids as red dots. You can see why they would be less accurate than points geocoded to street level. (The centroid positions are delivery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



### Geographic Geocoding

If the input address includes a valid combination of city, county, and state (but no further address information), you can still geocode to the city, county, or state centroid. MapMarker returns the most precise geographic centroid that it can based on the user input. Geographic geocoding is less precise than street or postal geocoding, but may be suitable for certain applications.

This image shows the geographic (city) centroids as blue circles.



### Building an Input Address

The input address is the address to be geocoded. MapMarker 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.

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

\*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.

Desired Geocoding Level	Required Input Fields	Other Requirements
Street	Street and ZIP Code, or street, city, and state	Country required in generic address structure.
Street Intersection	Street and ZIP Code, or street, city, and state  Street 1 and street 2 names must be separated by double ampersands "&&" (for example, "Global View && Jordan Road", "42nd && Park")	
Postal Centroid	ZIP Code™ and/or ZIP + 4®	Country required in generic address structure
Geographic	City, County and State	Available through Java API

## Using Preferences

As explained in [Geocoding Preferences on page 39](#), 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.

MapMarker 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. There are some USA-specific versions of preference names. The USA MustMatchStreet is equivalent to the generic MustMatchMainAddress, MustMatchCity is equivalent to the generic MustMatchAreaName3, and MustMatchZipcode is equivalent to the generic MustMatchPostalCode.

### 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.

Locate	Address	PostCode	Town	Region	ResultCode	Close
<a href="#">Map</a>	350 OCEAN DR	33149	KEY BISCAIYNE	FL	S5HPNTSC-A	true
<a href="#">Map</a>	350 OCEAN BLVD	33160	GOLDEN BEACH	FL	S2HPN-S-A	false
<a href="#">Map</a>	1800 S OCEAN DR	33160	NORTH MIAMI BEACH	FL	S5-NTS-A	false
<a href="#">Map</a>	9200 OCEAN CURVE DR	33189	MIAMI	FL	S5-P-TS-A	false
<a href="#">Map</a>	9200 OCEAN CURV	33189	MIAMI	FL	S5-PN-S-A	false

#### **Matching on Address Number; Close Matches Only**

When you 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.

Locate	Address	PostCode	Town	Region	ResultCode	Close
<a href="#">Map</a>	350 OCEAN DR	33149	KEY BISCAIYNE	FL	S5HPNTSC-A	true

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

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

3050 Ocean Drive  
Key Biscayne, FL 33149

returns a postal candidate (Z1 result code) 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 used and a postal candidate is returned.

Locate	Address	PostCode	Town	Region	ResultCode	Close
<a href="#">Map</a>		33149	----		Z1	true

**Matching on Address Number, Close Match Only, Falling Back to Geographic**

When you enable Close Matches Only, Must Match Address Number, and Fallback to Geographic preferences, the input address:

55 Upland Dr.  
Ithaca, NY

returns Ithaca, NY as a close geographic match (G3 result code).

Only a geographic candidate is returned because the Must Match Address Number preference is enabled and there is no exact address match on 55 Upland Dr.

City	State	Zip	GeoResult
ITHACA	NY	14850	G3

**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, 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, see the Result Codes chapter in the MapMarker Plus Desktop User Guide.

### Geocode Type

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

- S – Street Type
- Z – ZIP Code™ or postal code type
- M – multiple match
- G – geographic type
- N – non-match

### Positional Accuracy

The second character in the code reflects the positional accuracy of the candidate's point. The character can be the numbers 0-8, or the letter X.

- 8 – matched to a known addresses with exact latitude/longitude coordinates for each address.
- 7 – matched to an interpolated point along the candidate's street segment.
- 6 – matched to a point ZIP centroid. A point ZIP is a 5-digit ZIP Code™ that represents P.O. Box™ ZIPs and other unique ZIPs such as a single site, building, or organization.
- 5 – Street interpolated point
- 4 – Street centroid point
- 3,2,1 – Postal centroid point
- 0 – No geometry available (extremely rare)
- X – Intersection point

## Address Elements

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

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

## Unmatched Elements in Input Address

A dash "-" in the result code indicates that this element of the input address could not be matched. For example, in the result code: S5HPNTSC-A the dash appears instead of a Z. This means that the ZIP Code was not matched.

## Optimizing DPV and LACSLink Performance

DPV® and LACSLink® geocoding provide address standardization and validation capabilities, and are described more fully in the *MapMarker Plus User Guide*. If you use either of these features, the additional standardization and validation does impact MapMarker Plus performance. However, you can take steps to optimize DPV and LACSLink performance.

For the Server product, you can configure the USA\_DataManagerSettings.properties file to optimize DPV® and LACSLink®. There are two instances of the USA\_DataManagerSettings.properties file that you must consider: Look for this properties file in the following locations (<install> represents your MapMarker installation directory)

Properties File Location	Use and Configuration
<install>\sdk\engine\lib\client\	This instance of the USA_DataManagerSettings.properties file is present if you installed the Developer Kit (SDK). If you developed your own Java Application, (or are using the provided MapMarker Sample Application), you can configure this file to optimize DPV and LACSLink performance.
<install>\sdk\tomcat\webapps\mapmarker40\WEB-INF\classes\	This instance of the USA_DataManagerSettings.properties file is used by the tomcat web server. You can configure this file to optimize DPV and LACSLink performance on the server.

Add the following entries to `USA_DataManagerSettings.properties` file optimize DPV and LACSLink performance:

```
DPVOptimize=true
LACSLinkOptimize=true
```

When set to true, these entries allow map the DPV and LACSLink data files to be mapped into memory, and thereby improve the speed of a DPV or LACSLink geocoding. When set to false, no such memory mapping or optimization takes place. If these entries are absent, then no optimization takes place.

**Note** If your application will not use DPV or LACSLink features, do implement DPV or LACSLink optimization. This would have no benefit and would consume some system resources.

There is another instance of the `USA_DataManagerSettings.properties` file in the WAR file. If you deploy your application under a different web server that originally deployed (such as another tomcat or WebSphere), you must make these same configuration changes in the `USA_DataManagerSettings.properties` file that is packaged inside the WAR file. This instance of `USA_DataManagerSettings.properties` is located in:

```
<install>\sdk\tomcat\wars\mapmarker40.war
```

**Note** For the Desktop Application, DPV and LACSLink optimization is controlled in the System Preferences dialog. This is described in the *MapMarker Plus User Guide*. The Desktop Application ignores the optimization entries in `USA_DataManagerSettings.properties`.

# 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** .....50
- ♦ **Client/Server Geocoding** .....52
- ♦ **MapMarker Server** .....53
- ♦ **Running MapMarker Server on Windows Systems**.....53
- ♦ **Compiling and Running Existing Applications**. ....55

## MapMarker Geocoding API Development

Pitney Bowes 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 built on Java engines and use a full Java API, we have retired 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, see [Client/Server Geocoding on page 52](#) and [OLE Automation API in Chapter 5 on page 57](#).

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 55](#).

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 had to use the MapMarker J Server. Because both the MapMarker USA Java API and Envinsa Java API provide full geocoding functionality and communicate directly with the Java engine, these APIs eliminate the need for the older J Server API.

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

Java API	Description	Status
J Server 2.x API	A servlet that uses JNI to access the C-based geocoding engine. J Server clients send requests via HTTP.	This API is no longer supported.
J Server 3.x API	A servlet that uses JNI to access the C-based geocoding engine. J Server clients send requests via HTTP.	Deprecated

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 193](#).

## Windows Developers

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

Windows API	Description
OLE Automation API	The OLE Automation API is used to create customized OCX geocoding applications.
RPC Server API	The MapMarker RPC API is for the C developer who wants to create geocoding applications that call the MapMarker RPC Server.
GeoEng API	This is the C API for the MapMarker geocoding engine used for creating batch geocoding applications. The MapMarker geocoding engine and its API are packaged as a 32-bit Dynamic Link Library.

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 limited choices if they plan to use the C API. 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 for the C API for non-Windows developers. The following table provides a description and status of the APIs available for non-Windows developers.

API	Description
GeoEng API	This is the C API for the MapMarker geocoding engine used for creating batch geocoding applications. The MapMarker geocoding engine and its API are packaged as a shared library or shared object.
RPC Server API	The MapMarker RPC API is for the C developer who wants to create geocoding applications that call MapMarker RPC (Noble Net) Server.

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 Java API.

## Database Developers

Developers who want to include geocoding capabilities in their applications can choose the appropriate database product. The table below describes each one and gives its status.

Product	Description	Status
Informix DataBlades	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.	Support for this product has been retired.
MapMarker Geocoding Extender for Oracle	The Oracle Geocoding Extender product is a thin, pass-through layer on top of the Java client classes, which provide a SQL interface for address geocoding to MapMarker Java server.	The Geocoding Cartridge includes MapMarker USA functionality.
MapInfo Geocoding Extender for SQL Server 2005	The SQL Server 2005 Extender product is for SQL server database programmers. This product is a thin, pass-through layer on top of a .NET client which provides an SQL interface for address geocoding via XML to the MapMarker Java Server or Envinsa Location Utility service	The Geocoding Extender includes MapMarker USA functionality.
ESP SQL Server 2000	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.	The ESP SQL Server includes MapMarker USA functionality.

## 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 must install MapMarker as a Windows Service manually:

- At the command 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 command 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. 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.

[Configuring as a Windows Service, p. 53](#)

[Configuring as a Console Application, p. 54](#)

[Geocoding Request Timed Out, p. 55](#)

### Configuring as a Windows Service

To configure MapMarker Server as a Windows Service:

1. From Control Panel > Administrative Tools in Windows 2000, 2003, or XP, click **Services**. The Services dialog box is displayed.
2. Highlight the service called MapMarker Server.
3. Choose the **Startup** button. Configure the MapMarker Server to run as the user who installed MapMarker. Be sure the user has permission to start a service at startup time. Click **OK** to return to the Services dialog box.

4. Optional: To set the server for geocoding to a custom user dictionary, in the Start Parameters list box add `-udonly`. MapMarker does not initialize the MapMarker Address Dictionary. It gets match candidates from the user dictionary only.
5. Optional: To set the server for geocoding to ZIP Code™ centroids, in the Start Parameters list add `-ziponly`. MapMarker Server only geocodes to ZIP Code centroids; it does not return street address candidates.

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

6. Click **Start**. The MapMarker Server service starts.
7. To leave the Services dialog box, choose **Close**.

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

When the application is started, the server sends a message indicating the data path that was used when the MapMarker engine initialized. 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 Plus\USA\<version number>\SYSTEM`.

**Note** In references to the MapMarker registry keys, `<version number>` indicates the major-release version of MapMarker (14.0), even if you are running one of the point releases of that version.

If you encounter a problem with the Name Service Provider that prevents MapMarker Server from running when your machine boots, modify the Windows Services as follows:

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

To stop the MapMarker service:

1. Choose Control Panel > Administrative Tools > Services and highlight MapMarker Server.
2. Click **Stop**.

To remove the MapMarker service from Windows Services:

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

## Configuring as a Console Application

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

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

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

When the application is started, the server sends a message indicating the data path that was used when the MapMarker engine initialized. 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 Plus\USA\<version number>\SYSTEM`.

**Note** In references to the MapMarker registry keys, `<version number>` indicates the major-release version of MapMarker (14.0), even if you are running one of the point releases of that version.

To stop the application:

1. From the console window, press Ctrl-C.
2. When the command 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, (that is., 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. If that happens, 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 use ESP SQL Server should use the Windows RPC Server with MapMarker 10.x or higher.

## 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\geoen\include folder where MapMarker is installed.

```
#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, see [JNI Adapter Configuration File Settings in Appendix A on page 193](#).

# OLE Automation API

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:

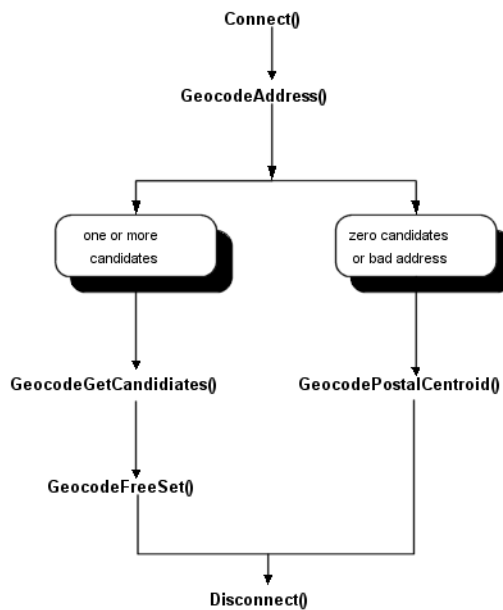
- ♦ **Creating a Custom Geocoding Control** .....58
- ♦ **Geocoder Control Properties, Events, and Methods** .....59
- ♦ **OLE Automation Methods** .....64
- ♦ **OLE Automation Objects** .....103
- ♦ **Programming Usage Example** .....105

## 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 Plus you must use the OLE Automation API. Calls are passed either directly through the JNI adapter to the MapMarker USA Java API, or to the RPC Server, and then through the JNI adapter to the MapMarker USA 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 are relevant when developing a custom geocoding application using the MapMarker Geocoder Control and/or the OLE Automation API.

### Input Properties

Input Property Name	Description
Firm	String: Firm name
Street	String: Street Address
Street2	String: Secondary Address
City	String: City
State	String: State
Zip	String: ZIP Code™
ZipPlus4	String: ZIP Code add-on
ServerName	String: Name of Server where MapMarker Server is running
NotUsingServer	Boolean: If TRUE does not connect to the server. Place mapmarkr.ocx in the same directory as the geengine mm32v_.dll. Default: False.
BCloseCandidates	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.
ExactHouse	Boolean: If TRUE, exact match on house number is required Default: True.
ExactName	Boolean: If TRUE, exact match on street name is required Default: False.
ExactZIP	Boolean: If TRUE, exact match on ZIP Code is required Default: False.
ExactCity	Boolean: If TRUE, exact match on City is required. Default: False
ExpandSearch	Boolean: If TRUE, MapMarker expands the search area in which it looks for match candidates. Default: False.
ExpandSearchInState	Boolean: If TRUE, the expanded search area is limited to the state. Default: False.

**Input Properties (continued)**

Input Property Name	Description
ExpandDistance	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.
MatchIntersections	Boolean: If TRUE, street intersection matching is attempted. Default: False.
PreferUserDictionary	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.
LinearOffset	Double: Defines the position of the geocoded point with respect to the corner. Default: 25.
PerpendicularSetback	Double: Defines the position of the geocoded point with respect to the centerline of the street. Default: 20.
Units	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
ShowPropertiesButton	Boolean: If TRUE, the Show Properties button is visible. Default: True.

**Input Properties (continued)**

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

**Output Properties**

<b>Output Property Name</b>	<b>Description</b>
Latitude	Double: The latitude value for a match candidate.
Longitude	Double: the longitude value for a match candidate.
Precision	Integer: A number that identifies match precision (street level, shape path, intersection, point ZIP, or ZIP centroid).  No centroid 0 ZIP Code™ 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.

Output Properties

Output Property Name	Description
DatabaseTypes	Integer: a number that identifies the available databases: 2 - ZIP Code™ centroid database 3 - Street and ZIP Code databases 4 - User Dictionary 6 - User Dictionary and ZIP Code databases 7 - Street, User Dictionary, and ZIP Code databases
StringBinding	String: RPC binding string used to connect to the server.
PrimaryStreet	String: The candidate primary street, if it has one.
RecordType	String: The candidate ZIP + 4® record type as categorized by the U.S. Postal Service®. If the match is from a TIGER record, the candidate may not have a record type.  F – Firm G – General Delivery H – Highrise P – PO Box R – Rural Route/Highway S – Street
DeliveryPoint	String: The candidate delivery point.
Carrier Route	String: The candidate carrier route.
CheckDigit	String: The candidate check digit.

## Output Properties

Output Property Name	Description
TabbedAddress	<p>String: The candidate street address components delimited by tabs as follows:</p> <ul style="list-style-type: none"> <li>House Number</li> <li>Directional Prefix</li> <li>Street Type Prefix</li> <li>Street Name</li> <li>Street Type Suffix</li> <li>Directional Suffix</li> <li>Unit Type</li> <li>Unit Value</li> </ul> <p>Example: 271 E Deering Ave Apt 3.</p>
Lacs	<p>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<sup>®</sup> product, known as the Locatable Address Conversion Service.</p>
Pmb	<p>String: Private Mailbox used at Commercial Mail Receiving Agencies (CMRA) such as Mailboxes Etc.</p>
PmbRange	<p>String: The candidate Pmb range/number when there is a PMB range in the input.</p>
AddressType	<p>Integer. MapMarker address type.</p> <ul style="list-style-type: none"> <li>Street 10</li> <li>Place 11</li> <li>ZIP 12</li> <li>Rural 13</li> <li>Highway 14</li> <li>PO Box 15</li> <li>Military 16</li> <li>Intersection 17</li> </ul>

## Events

Event	Purpose
LatLongChanged()	This event is triggered when the Geocode button is pressed, the <code>DoGeocode()</code> 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.
GeocodeEvent()	This event is triggered when the Geocode button is pressed and the whole geocoding process is completed.  <b>Note</b> The <code>DoGeocode()</code> method does not trigger this event.

## OLE Automation Methods

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

- ◆ **ClearDialogText()** ..... 66  
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()** ..... 67  
Builds an RPC binding string to connect to the MapMarker Server. Calls `GeocodeCheckDbAvailability()`.
- ◆ **Disconnect()** ..... 68  
Disconnects from the MapMarker Server.
- ◆ **DoGeocode()** ..... 68  
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()** ..... 68  
Brings up a set of property pages for the Geocoder Control.
- ◆ **DpvGeocodeAddress()** ..... 69  
Turns DPV mode on, and then geocodes an address and builds a list of candidates.
- ◆ **DpvGeocodeAddressLastLine()** ..... 70  
Turns DPV mode on, and then allows you to enter an address with an unparsed last line (for example, the city, state, and/or ZIP Code are contained in the same field in the database).
- ◆ **DpvGeocodeAddressWithSerial()** ..... 73  
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()** ..... 75  
Attempts to geocode an address and build a list of candidates.
- ◆ **GeocodeAddressEx()** ..... 78  
Attempts to geocode an address and build a list of candidates. Also enables street2 information to be passed in as a separate field.

- ◆ **GeocodeAddressLastLine()** . . . . . 79  
Enables you to enter an input address with an unparsed line containing the city, state, and/or ZIP Code.
- ◆ **GeocodeAddressLastlineEx()** . . . . . 81  
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()** . . . . . 82  
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()** . . . . . 83  
Requires user to input a serial number for geocoding.
- ◆ **GeocodeAddressWithSerialEx()** . . . . . 84  
Requires user to input a serial number for geocoding, and enables street2 information to be passed in as a separate field.
- ◆ **GeocodeCheckDbAvailability()** . . . . . 86  
Checks to see which types of database(s) are available for geocoding.
- ◆ **GeocodeFreeSet()** . . . . . 86  
Frees the server side information after geocoding.
- ◆ **GeocodeGetCandidates()** . . . . . 87  
Gets all the candidate information after geocoding.
- ◆ **GeocodeGetErrorText()** . . . . . 88  
Returns a short text description of the specified error.
- ◆ **GeocodeGetHwyExitCandidate()** . . . . . 88  
Gets the specified highway exit candidate after a call to GeocodeHwyExit\_A().
- ◆ **GeocodeGetServerVersion()** . . . . . 90  
Returns the server version number.
- ◆ **GeocodeGetStatesFound()** . . . . . 90  
Returns a list of 2-digit state abbreviations representing states found in the Address Dictionary path. These are not necessarily licensed states.
- ◆ **GeocodeGetStatesLicensed()** . . . . . 91  
Returns a list of 2-digit state abbreviations representing licensed states.
- ◆ **GeocodeHwyExit()** . . . . . 91  
Returns a count of the number of highway records based on input criteria of highway exit and state.
- ◆ **GeocodelsCandidateMultiUnit()** . . . . . 93  
Returns a True/False value, indicating whether the candidate address contains multiple units.
- ◆ **GeocodePostalCentroid()** . . . . . 93  
Geocodes a ZIP Code or ZIP + 4 centroid.
- ◆ **GeocodePostalCentroidWithSerial()** . . . . . 94  
Requires a serial number for ZIP Code-level geocoding.
- ◆ **GetCandidateAt()** . . . . . 95  
Returns the line of text in the Match Candidate list box.
- ◆ **GetCandidateCensusBlockIDAt()** . . . . . 96  
Returns the Census Block ID as a string.
- ◆ **GetCandidateCityAt()** . . . . . 96  
Returns the city portion of the address as a string.

## ClearDialogText()

---

- ◆ **GetCandidateFirmAt()** . . . . . 97  
Returns the firm portion of the address as a string.
- ◆ **GetCandidateLatitudeAt()** . . . . . 97  
Returns the latitude as a double.
- ◆ **GetCandidateLongitudeAt()** . . . . . 98  
Returns the longitude as a double.
- ◆ **GetCandidatePlus4At()** . . . . . 98  
Returns the ZIP add on of the address as a string.
- ◆ **GetCandidatePrecisionAt()** . . . . . 99  
Returns the precision for the match as a string. Precision refers to the quality of the match: to street level, shape path, intersection, point ZIP, or ZIP centroid.
- ◆ **GetCandidateResultCodeAt()** . . . . . 99  
Returns the result code portion of the address as a string.
- ◆ **GetCandidateStateAt()** . . . . . 100  
Returns the state portion of the address as a string.
- ◆ **GetCandidateStreetAt()** . . . . . 100  
Returns the street portion of the address as a string.
- ◆ **GetCandidateZIPAt()** . . . . . 101  
Returns the ZIP portion of the address as a string.
- ◆ **GetFullName()** . . . . . 101  
Returns the full name and path of the application.
- ◆ **GetName()** . . . . . 102  
Returns the application name.
- ◆ **GetVersionNum()** . . . . . 102  
Returns the version number of the application.
- ◆ **RefreshDialog()** . . . . . 102  
Forces the dialog box to repaint.
- ◆ **SelectCandidateAt()** . . . . . 102  
Highlights the specified candidate in the Match Candidate List box.

---

## ClearDialogText()

### Purpose

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

### Syntax

```
ClearDialogText ()
```

### Returns

The 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

Parameter	Description
<i>NetworkAddress</i>	[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.

### Returns

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

Return	Return #	Description
GEO_ENG_BAD_PARAM_ERR	3	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	2	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	10	Error accessing the Address Dictionary.
GEO_ENG_MISSING_DATABASE_ERR	1078	Address Dictionary file(s) not found.
GEO_ENG_ZIP_MASTER_FILE_NOT_FOUND_ERR	1228	Zipmastr.cdb or Zipmastr.jdx not found.
GEO_ENG_DBQ_MISSING_LICFILE	1079	License file not found.
GEO_ENG_INVALID_SERIAL_NUMBER	1085	Invalid serial number.
GEO_ENG_PARSE_INIT_ERR	307	Parsing file (geo_usa.*) missing or bad.

## Disconnect()

---

Return	Return #	Description
GEO_ENG_MATCH_INIT_ERR	563	Matching file (geo_usa.*) missing or bad.
GEO_ENG_NADCON_MISSING_FILE	1334	*.las/*.los files not found.

---

## Disconnect()

### Purpose

This call is used to disconnect from the MapMarker Server.

### Syntax

```
Disconnect() As Boolean
```

### Returns

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

## DoGeocode()

### Purpose

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

### Syntax

```
DoGeocode()
```

## DoSetProperties()

### Purpose

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

### Syntax

```
DoSetProperties() As Boolean
```

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

## DpvGeocodeAddress()

### Purpose

This call turns DPV® mode on, and then geocodes an address and builds a list of candidates. It also enables you to pass street2 information as a separate field. 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 USA.

### 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

Parameter	Description
geocodeHandle	[Output Long] Used to reference information about the geocoding operation in other calls.
firm	[Input String] Firm name to be matched.
street	[Input String] Street name to be matched.
street2	[Input String] Secondary address.
city	[Input String] City name to be matched.
state	[Input String] State to be matched.
zip	[Input String] Postal code to be matched.

## DpvGeocodeAddressLastLine()

---

Parameter	Description
<code>status</code>	[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.
<code>numCandidates</code>	[Output Integer] Total number of candidates found.
<code>numCloseCandidates</code>	[Output Integer] 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.

---

## DpvGeocodeAddressLastLine()

### Purpose

This method turns DPV® mode on, and then allows you to enter an address with an unparsed last line (for example, 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 USA.

## Syntax

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

## Parameters

Parameter	Description
geocodeHandle	[Output Long] Used to reference information about the geocoding operation in other calls.
firm	[Input String] Firm name to be matched.
street	[Input String] Street name to be matched.
street2	[Input String] Secondary address.
lastline	[Input String] Unparsed last line of address to be matched.

## 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 (for example, 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 be used instead of GeocodeAddressLastline, which used to have an optional serial number parameter.

**Note** CASS™ and DPV functionality are available with MapMarker USA.

## Syntax

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

## Parameters

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

## 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<sup>®</sup> 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<sup>™</sup> and DPV functionality are available with MapMarker USA.

### 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

Parameter	Description
geocodeHandle	[Output Long] Used to reference information about the geocoding operation in other calls.
firm	[Input String] Firm name to be matched.
street	[Input String] Street name to be matched.
street2	[Input String] Secondary address.
city	[Input String] City name to be matched.
state	[Input String] State to be matched.
zip	[Input String] Postal code to be matched.

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

### Returns

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

## GeocodeAddress()

### Purpose

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

### Syntax

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

### Parameters

Parameter	Description
geocodeHandle	[Output Long] Used to reference information about the geocoding operation in other calls.
firm	[Input String] Firm name to be matched.
street	[Input String] Street name to be matched.
city	[Input String] City name to be matched.
state	[Input String] State to be matched.
zip	[Input String] Postal code to be matched.

Parameter	Description
status	<p>[Output Integer] One of the following:</p> <p>-1 – UNIQUE_ZIP_NO_MATCH – no match to record containing unique ZIP Code (CASS™ mode only).</p> <p>0 – SINGLE_MATCH – a single close match was found for the input address.</p> <p>1 – MULTIPLE_MATCH – multiple match candidates were found and MapMarker could not choose between at least two of them.</p> <p>2 – NO_MATCHES – candidates found, but none considered a close match.</p> <p>3 – NO_CANDIDATES – no candidates found for the input address.</p> <p>4 – SINGLE_INTERSECT_MATCH – an intersection match was found.</p> <p>5 – MULTIPLE_INTERSECT_MATCH – more than one intersection candidate was found.</p> <p>6 – NO_INTERSECT_MATCHES – no close intersection candidates found.</p> <p>7 – NO_INTERSECT_CANDIDATES – no intersection candidates found.</p> <p>8 – POSSIBLE_INTERSECTION – possible close intersection match found.</p>
numCandidates	[Output Integer] Total number of candidates found.
numCloseCandidates	[Output Integer] 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.

### 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 105](#), 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

Parameter	Description
geocodeHandle	[Output Long] Used to reference information about the geocoding operation in other calls.
Firm	[Input String] Firm name.
Street	[Input String] Street name.
Street2	[Input String] Secondary address.
City	[Input String] City.
State	[Input String] State.
Zip	[Input String] ZIP Code™ (includes ZIP + 4®).
status	[Output Integer] The match status of the geocode. See <a href="#">GeocodeAddress() on page 75</a> for a full description of the status parameter.
numCandidates	[Output Integer] The total number of candidates.
numCloseCandidates	[Output Integer] The number of close match candidates.

## 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 (for example, the city, state, and/or ZIP Code are contained in the same field in the database).

### Syntax

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

**Parameters**

<b>Parameter</b>	<b>Description</b>
<code>geocodeHandle</code>	[Output Long] Used to reference information about the geocoding operation in other calls.
<code>firm</code>	[Input String] Firm name to be matched.
<code>street</code>	[Input String] Street name to be matched.
<code>lastline</code>	[Input String] Unparsed last line of address to be matched.
<code>status</code>	[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.
<code>numCandidates</code>	[Output Integer] The total number of candidates found.
<code>numCloseCandidates</code>	[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.

## GeocodeAddressLastlineEx()

### Purpose

This method allows you to geocode an address using an unparsed last line (for example, 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

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

### 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 (for example, 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 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

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

### Returns

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

## GeocodeAddressWithSerial()

### Purpose

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

### Syntax

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

### Parameters

Parameter	Description
geocodeHandle	[Output Long] Used to reference information about the geocoding operation in other calls.
firm	[Input String] Firm name to be matched.
street	[Input String] Street name to be matched.
city	[Input String] City name to be matched.
state	[Input String] State to be matched.
zip	[Input String] Postal code to be matched.

## GeocodeAddressWithSerialEx()

---

Parameter	Description
<code>status</code>	[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.
<code>numCandidates</code>	[Output Integer] The total number of candidates found.
<code>numCloseCandidates</code>	[Output Integer] The number of close candidates.
<code>SerialNumber</code>	[Input String] Serial number required by server, created by an OEM license program sold separately.

### Returns

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

---

## GeocodeAddressWithSerialEx()

### Purpose

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

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

## Syntax

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

## Parameters

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

## 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

Parameter	Description
<code>geocodeHandle</code>	[Input/Output] Used to reference information about the geocoding operation in other calls.

**Returns**

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

**GeocodeGetCandidates()****Purpose**

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

**Syntax**

```
GeocodeGetCandidates(geocodeHandle As Long,
                    numCandidates As Integer) As Object
```

**Parameters**

Parameter	Description
<code>geocodeHandle</code>	[Input Long] Used to reference information about the most recent geocoding operation.
<code>numCandidates</code>	[Input Integer] Specifies the number of candidates that should be returned (for example, the size of the collection).

**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 105](#), 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
```

## 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

Parameter	Description
<code>errorCode</code>	[Input Integer] MapMarker error code.
<code>errorString</code>	[Output String] Error text.

### 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

Parameter	Description
index	[Input Integer] Zero-based index of candidate to retrieve.
HwyTypePre	[Output String] Type Prefix of the matched highway (for example, Highway 21).
HwyName	[Output String] Name of the matched highway.
HwyTypeSuf	[Output String] Type Suffix of the matched highway (for example, Pennsylvania Turnpike).
HwyDir	[Output String] The direction of the matched highway.
ExitNum	[Output String] The matched exit number.
ExitName	[Output String] The matched exit name.
City	[Output String] The returned city name.
State	[Output String] The returned state name.
Zipcode	[Output String] The returned ZIP Code™.
Longitude	[Output Double] The X coordinate of the returned point.
Latitude	[Output Double] The Y coordinate of the returned point.

## 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

Parameter	Description
<code>pVersionNum</code>	[Output Double] MapMarker version in integer format (for example, 13)

### 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

Parameter	Description
<code>pstateList</code>	[Output String] A list of space-separated 2-digit state abbreviations.

### 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

Parameter	Description
<code>pstateList</code>	[Output String] A list of space-separated 2-digit state abbreviations representing licensed states.

### 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**

<b>Parameter</b>	<b>Description</b>
Hwyexit	<p>[Input String] The highway and exit information in the form:</p> <p>&lt;highway name&gt; &lt;highway directional&gt; EXIT&lt;exit number&gt;&lt;exit number suffix&gt;TO +&lt;exit name&gt;. Where:</p> <p>highway name is the name of the highway name including type prefix or type suffix (for example, Highway 8, I-90, or Pennsylvania Turnpike).</p> <p>highway directional is the direction of the highway, full or abbreviated (for example, E or East). Optional.</p> <p>EXIT keyword is a case-insensitive keyword used to separate the highway information from the exit information. Required.</p> <p>exit number is the actual number of the exit. This is optional if there is an exit name in the input.</p> <p>exit number suffix is any non-numerical exit suffix (for example, Exit 2E). Optional, not used by MapMarker.</p> <p>TO keyword is a case-insensitive keyword used to separate the exit name and exit number. Optional.</p> <p>exit name is the name of the exit (for example, 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.</p> <p>Sample input:</p> <p>I-87 EXIT 2E TO State Hwy 5</p> <p>I-87 EXIT State Hwy 5</p> <p>I-87 EXIT 2E</p> <p>I-87 EXIT 2</p> <p>I-87 EXIT 2E State Hwy</p> <p>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).</p>
State	[Input String] Contains the state name, either in full or abbreviated.
count	[Output Integer] The total number of HwyExit candidates that matched.

**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.

## 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

Parameter	Description
<code>geocodeHandle</code>	[Input] The handle returned from the Geocode call used.
<code>index</code>	[Input] Index value of the candidate to examine. This index is zero-based.
<code>IsMultiUnit</code>	[Output] True/False value indicating whether the candidate address has multiple units.

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
```

## GeocodePostalCentroidWithSerial()

---

### Parameters

Parameter	Description
<i>Zip</i>	[Input String] ZIP Code™ address component
<i>Plus4</i>	[Input String] ZIP Code add-on address component
<i>Longitude</i>	[Output Double] Returns the longitude value.
<i>Latitude</i>	[Output Double] Returns the latitude value.
<i>Precision</i>	[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).
<i>ResultCode</i>	[Output String] Returns a value that is analogous to the GeoResult codes returned by MapMarker.

### 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 PBSI.

### Syntax

```
GeocodePostalCentroidWithSerial(Zip As String,  
    Plus4 As String,  
    Longitude As Double,  
    Latitude As Double,  
    Precision As Integer,  
    ResultCode As String,  
    SerialNumber As String) As Boolean
```

## Parameters

Parameter	Description
Zip	[Input String] ZIP Code™ address component
Plus4	[Input String] ZIP Code add-on address component
Longitude	[Output Double] Returns the longitude value.
Latitude	[Output Double] Returns the latitude value.
Precision	[Output Integer] Returns one of the following: 10 (Shape Path Centroid match); 20 (street-level match for street address); 30 (street-level match for an intersection address); 40 (point ZIP) 1 (ZIP Code Centroid match). 2 (ZIP + 2 Centroid match); or 3 (ZIP + 4® Centroid match);
ResultCode	[Output String] Returns a value that is analogous to the Geosresult codes returned by MapMarker.
SerialNumber	[Input String] Serial number required by server, created by an OEM license program sold separately.

## 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

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

### 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

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

### Returns

The Census Block ID of the specified candidate. The `LastErrorCode` property can be used to determine the nature of any errors that occur.

---

## GetCandidateCityAt()

### Purpose

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

### Syntax

```
GetCandidateCityAt (Index As Integer) As String
```

**Parameters**

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

**Returns**

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

**GetCandidateFirmAt()****Purpose**

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

**Syntax**

```
GetCandidateFirmAt(Index As Integer) As String
```

**Parameters**

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

**Returns**

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

**GetCandidateLatitudeAt()****Purpose**

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

**Syntax**

```
GetCandidateLatitudeAt(Index As Integer) As Double
```

## GetCandidateLongitudeAt()

---

### Parameters

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

### 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

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

### 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**

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

**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<sup>®</sup>, ZIP + 2 or ZIP Code<sup>™</sup>).

**Syntax**

```
GetCandidatePrecisionAt(Index As Integer) As String
```

**Parameters**

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

**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

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

### Returns

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

---

## GetCandidateStateAt()

### Purpose

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

### Syntax

```
GetCandidateStateAt (Index As Integer) As String
```

### Parameters

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

### Returns

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

---

## GetCandidateStreetAt()

### Purpose

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

### Syntax

```
GetCandidateStreetAt (Index As Integer) As String
```

**Parameters**

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

**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**

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

**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()

---

### 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

Parameter	Description
Index	[Input Integer] The number from the zero-based index list that corresponds to the desired candidate.

## 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<sup>®</sup> 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<sup>®</sup> 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<sup>®</sup> 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<sup>®</sup> file.
- A1—Input address not matched to the ZIP + 4 file.
- BB—Input address matched to DPV<sup>®</sup> (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.

The screenshot shows a Visual Basic form titled "Form1" with a standard Windows-style title bar. The form is organized into several functional areas:

- Server Configuration:** A group box "Server or Local" contains two radio buttons: "Use MM Server" (selected) and "Use Local DLL". To the right are fields for "Server Name", "Database Types", "Database Found", "Binding", and "Version", along with "Connect" and "Disconnect" buttons. Below these is an "Executable Name" field containing "VB6.EXE".
- Address Input:** A series of text boxes for "Firm", "Street", "City", "State", "ZIP", "ZIP+4", "Longitude", "Latitude", and "Precision".
- Geocoding Actions:** Buttons for "Geocode", "Geocode Centroid", "GetCandidates", and "FreeSet".
- Results:** A "Candidates:" label followed by a list box containing "[firm, ]street, city, state zip-zip+4 (coordinates) Precision CensusBlock".
- Search Options:**
  - Match On:** Checkboxes for "House Number" (checked), "Street Name", and "ZIP Code".
  - Extended Search:** Checkboxes for "Extended Search" and "In Current State".
  - Offset:** Numeric fields for "Offset from Road" (25) and "Offset from Center" (20), plus a "Match Intersections" checkbox.
- Additional Settings:** "Licensed For:" and "Available States:" text boxes, each with a "Set" or "Refresh" button.

## Programming Usage Example

---

```
Dim objMM As Object 'MapMarkr
Dim lngEngHandle&
Dim status As Integer
Dim numCandidates As Integer
Dim numCloseCandidates As Integer
Dim bConnected As Boolean
Dim bSettingModified As Boolean

Private Sub chkExtendedSearch_Click()
    bSettingModified = True
End Sub

Private Sub chkInState_Click()
    bSettingModified = True
End Sub

Private Sub chklastline_Click()
If chklastline.Value = 1 Then
    txtlastline.Visible = True
    Label21.Visible = True
    txtCity.Visible = False
    txtState.Visible = False
    txtZip.Visible = False
    txtZip4.Visible = False
    Label5.Visible = False
    Label6.Visible = False
    Label7.Visible = False
    Label8.Visible = False
Else
    txtlastline.Visible = False
    Label21.Visible = False
    txtCity.Visible = True
    txtState.Visible = True
    txtZip.Visible = True
    txtZip4.Visible = True
    Label5.Visible = True
    Label6.Visible = True
    Label7.Visible = True
    Label8.Visible = True
End If

End Sub

Private Sub chkMatchCentroid_Click()
    bSettingModified = True
End Sub

Private Sub chkMatchHouse_Click()
    bSettingModified = True
End Sub

Private Sub chkMatchStreet_Click()
    bSettingModified = True
```

```

End Sub

Private Sub chkMatchXsec_Click()
    bSettingModified = True
End Sub

Private Sub cmdCentroid_Click()
    Dim retVal As Boolean
    Dim dblLong#, dblLat#, intPrec%, strResult$
    lblStatus = ""
    lblCand = ""
    lstCandidates.Clear
    On Error GoTo GeocodePostalCentroid_Error
    retVal = objMM.GeocodePostalCentroid(txtZip, txtZip4, dblLong,
dblLat, intPrec, strResult)
    On Error GoTo 0
    If Not retVal Then
        errorCode = objMM.LastErrorCode
        If errorCode = 14 Then
            lblStatus = "Invalid address"
            'Sometimes .Connect returns true even when server is
            'not running, but .GeocodeAddress will return 1753
        ElseIf errorCode = 1753 Then
            If txtServer = "" Then
                lblStatus = "Local server is not running"
            Else
                lblStatus = "Server : " & txtServer & " is not
                running"
            End If
        Else
            lblStatus = "Error geocoding, error code: " &
                Str$(errorCode)
        End If
    Else
        lblLong = dblLong
        lblLat = dblLat
        Select Case intPrec
            Case 3
                lblPrec = "Zip+4 "
            Case 2
                lblPrec = "Zip+2 "
            Case 1
                lblPrec = "Zip centroid "
        End Select
    End If
    ' Centroid geocoding does not create candidates
    cmdGetCand.Enabled = False
Exit Sub
GeocodePostalCentroid_Error:
    MsgBox "Error Zip centroid geocoding"
End Sub

Private Sub cmdConnect_Click()

```

```
Dim retVal As Boolean
Dim serverVersion As Double
Dim txtBuf As String
    If bConnected = False Then
        On Error GoTo Connect_Error
        retVal = objMM.Connect(txtServer)
        txtDbAvailable = objMM.GeocodeCheckDbAvailability
        If retVal = True Then
            bConnected = True
            ' Only when connected, can do geocoding
            cmdGeocode.Enabled = True
            cmdCentroid.Enabled = True
            cmdDisconnect.Enabled = True
            cmdConnect.Enabled = False
            DatabaseTypes = objMM.DatabaseTypes
        Else
            DatabaseTypes = 0
        GoTo Connect_Error
        End If
        retVal = objMM.GeocodeGetServerVersion(serverVersion)
        txtVersion = Str$(serverVersion)
        retVal = objMM.GeocodeGetStatesLicensed(txtBuf)
        txtLicensed = txtBuf
        retVal = objMM.GeocodeGetStatesFound(txtBuf)
        txtAvailable = txtBuf
    End If
    cmdRefresh_Click
Exit Sub

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

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

```

```
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
    End With
End Sub
```

```

End If
If chkInState.Value = 1 Then
    .ExpandSearchInState = True
Else
    .ExpandSearchInState = False
End If
If chkMatchXsec.Value = 1 Then
    .MatchIntersections = True
Else
    .MatchIntersections = False
End If
.ExpandDistance = txtDistance
.LinearOffset = txtOffsetLine
.PerpendicularSetback = txtOffsetPerp
bSettingModified = False
End With
End Sub

Private Sub Form_Load()
    Set objMM = CreateObject("MAPMARKR.MapMarkrCtrl.1")
    bConnected = False
    bSettingModified = False
    cmdGeocode.Enabled = False
    cmdCentroid.Enabled = False
    cmdGetCand.Enabled = False
    cmdFreeSet.Enabled = False
    cmdDisconnect.Enabled = False
    txtlastline.Visible = False
    Label21.Visible = False

    With objMM
        txtServer = .ServerName
        lblBinding = .StringBinding
        ExeName = .GetName
        If .ExactHouse = True Then
            chkMatchHouse.Value = 1
        Else
            chkMatchHouse.Value = 0
        End If
        If .ExactName = True Then
            chkMatchStreet.Value = 1
        Else
            chkMatchStreet.Value = 0
        End If
        If .ExactZip = True Then
            chkMatchCentroid.Value = 1
        Else
            chkMatchCentroid.Value = 0
        End If
        If .ExpandSearch = True Then
            chkExtendedSearch.Value = 1
        Else
            chkExtendedSearch.Value = 0
        End If
    End With
End Sub

```

```
End If
If .ExpandSearchInState = True Then
    chkInState.Value = 1
Else
    chkInState.Value = 0
End If
If .MatchIntersections = True Then
    chkMatchXsec.Value = 1
Else
    chkMatchXsec.Value = 0
End If
txtDistance = .ExpandDistance
txtOffsetLine = .LinearOffset
txtOffsetPerp = .PerpendicularSetback
End With

End Sub

Private Sub Option1_Click()
    objMM.NotUsingServer = False
    cmdConnect.Enabled = True
    cmdGeocode.Enabled = False
    cmdCentroid.Enabled = False
    txtServer.Enabled = True
End Sub

Private Sub Option2_Click()
    Dim serverVersion As Double
    Dim txtBuf As String
    objMM.NotUsingServer = True
    cmdConnect.Enabled = False
    cmdGeocode.Enabled = True
    cmdCentroid.Enabled = True
    txtServer.Enabled = False
    txtDbAvailable = objMM.GeocodeCheckDbAvailability
    DatabaseTypes = objMM.DatabaseTypes
    retVal = objMM.GeocodeGetServerVersion(serverVersion)
    txtVersion = Str$(serverVersion)
    retVal = objMM.GeocodeGetStatesLicensed(txtBuf)
    txtLicensed = txtBuf
    retVal = objMM.GeocodeGetStatesFound(txtBuf)
    txtAvailable = txtBuf
    ExeName = objMM.GetName
End Sub

Private Sub txtDistance_Change()
    bSettingModified = True
End Sub

Private Sub txtOffsetLine_Change()
    bSettingModified = True
End Sub
```

```
Private Sub txtOffsetPerp_Change()  
    bSettingModified = True  
End Sub
```



# Using the MapMarker Java API

This chapter 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.

**Note** This chapter does not describe all of the MapMarker Java API methods. For complete coverage of API, refer to the API documentation (Javadocs). See [Using the JavaDocs API Documentation on page 118](#) for the location of the JavaDocs.

## In this chapter:

- ♦ [Using the MapMarker Java USA API](#) .....118
- ♦ [Using the MapMarker Generic Java API](#) .....118
- ♦ [Using the JavaDocs API Documentation](#) .....118
- ♦ [Developing Geocoding Applications in Java](#) .....118
- ♦ [Creating a Web Geocoding Client in Java](#) .....126
- ♦ [Browsing Addresses](#) .....131
- ♦ [Geocoding to Geographic Centroids](#) .....132
- ♦ [Geocoding to Highway Exits](#) .....134
- ♦ [Geocoding to Airports](#) .....135
- ♦ [Address Point Interpolation](#) .....136
- ♦ [Determining DPV Availability](#) .....137

## 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.

For complete API documentation, see, [Using the JavaDocs API Documentation](#).

## 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 172](#).

For complete API documentation, see [Using the JavaDocs API Documentation](#).

## Using the JavaDocs API Documentation

For detailed information on all API packages, interfaces, and classes, refer to the API documentation (Javadocs) for MapMarker.

For the generic Java API, the JavaDocs is located in `<install>\docs\core` folder, where `<install>` represents your MapMarker installation directory. From the core folder, run `index.html` to see the generic (core) MapMarker Javadocs. If you are running the Web Application, you can also browse to `http://localhost:8095` on a computer where the MapMarker Server is running. Then click on the link for the MapMarker Java API.

For the MapMarker Java USA API, the JavaDocs is located in `<install>\docs\usa` folder, where `<install>` represents your MapMarker installation directory. From the usa folder, run `index.html` to see the USA-specific MapMarker Javadocs. If you are running the Web Application, you can also browse to `http://localhost:8095` on a computer where the MapMarker Server is running. Then click on the link for the MapMarker USA v14 Java API.

## 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](#) and [Using the MapMarker Generic Java API](#) for more information on which API to use for your particular situation.

Make sure that your application contains the import statements below. Use one set or the other, depending on whether you are using the USA-specific API or the Generic Java API.

For the USA-specific API:

```
/* MapMarker 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;
```

For the Generic Java API

```
/* MapMarker Generic Java API */
import com.mapinfo.mapmarker.user.MMJEngine;
import com.mapinfo.mapmarker.user.ClientGeocodeResponse;
import com.mapinfo.mapmarker.user.GeocodeConstraints;
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.

```
/* 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 are using the MapMarker Generic API, use the Address class implemented with AddressImpl.

```
Address inpAddr = new AddressImpl();

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

## Geocoding Constraints

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.

The generic GeocodeConstraints are described in the following [GeocodeConstraints](#) table. The U.S.-specific constraints are described in [USA\\_GeocodeConstraints](#)

### GeocodeConstraints

Keys	Description	Default
KEY_CLIENT_CRCS	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.	"epsg:4326"
KEY_CLOSEMATCHESONLY	Only close matches are returned.	"false"
KEY_CORNEROFFSET	Defines the position of the geocoded point with respect to the corner.	"7"
KEY_CORNEROFFSETUNITS	The units used in KEY_CORNEROFFSET.	"m" (meters)
KEY_FALLBACK_TO_GEOGRAPHIC	If no close street level candidates are found, return geographic centroid for the input address.	"false"
KEY_FALLBACK_TO_POSTAL	If no close street level candidates are found, return postal centroid for the input address.	"false"
KEY_MAXCANDIDATES	Max candidates to be returned (use "-1" to return all candidates)	"3"

**GeocodeConstraints (continued)**

Keys	Description	Default
KEY_MAXRANGES	Max ranges per candidate to be returned (use "-1" to return all ranges)	"0"
KEY_MAXRANGEUNITS	Max units per range to be returned	"0"
KEY_MUST_MATCH_ADDRNUM	Only candidates matching address number are considered close.	"false"
KEY_MUST_MATCH_AREA1	Only candidates matching a country subdivision are considered close (for example, State for USA or Province for Canada).	"false"
KEY_MUST_MATCH_AREA3	Only candidates matching a city or town are considered close.	"false"
KEY_MUST_MATCH_INPUT	Only candidates matching all input areas (where MUST_MATCH is defined) are considered close.	"false"
KEY_MUST_MATCH_MAINADDR	Only candidates matching the main address are considered close.	"false"
KEY_MUST_MATCH_POSTAL	Only candidates matching postal code are considered close.	"false"
KEY_MATCH_MODE	This key contains the match mode value. This must be one of the defined match modes. The possible key values are described in <a href="#">Using Match Mode Constants on page 122</a> .	null (match mode is not implemented)
KEY_STREETOFFSET	Defines the position of the geocoded point with respect to the centerline of the street.	"7"
KEY_STREETOFFSETUNITS	The units used in KEY_STREETOFFSET.	"m" (meters)

### Using Match Mode Constants

The following constants are used with the **KEY\_MATCH\_MODE** constraint. The **CASS\_MODE** constant is used with **USA\_GeocodeConstraints** only.

**Match Mode Constants**

Constant	Value	Description
RELAXED_MODE	RelaxedMode	Sets Relaxed match mode.
EXACT_MODE	ExactMode	Sets Exact match mode.
DEFAULT_MODE	DefaultMode	Sets Default match mode
MATCH_MODE_NONE	null	Indicates that match mode is not used. This means that custom individual settings are used instead.
CASS_MODE (used with <b>USA_GeocodeConstraints</b> only)	CASSMODE	Sets CASS match mode. See sample code

### USA\_GeocodeConstraints

USA\_GeocodeConstraints extend the GeocodeConstraints object. This provides both a default and copy constructor (no arguments or a GeocodeConstraints object as the argument). The keys are defined in the following **USA\_GeocodeConstraints** table:

**USA\_GeocodeConstraints**

Keys and Constants	Base Equivalent	Description	Default
KEY_MUST_MATCH_ZIPCODE	POSTAL	Requires ZIP Code™ match for a close match.	"false"
KEY_MUST_MATCH_CITY	AREA3	Requires city/town match for a close match.	"false"
KEY_MUST_MATCH_STREETNAME	STREET	Requires street name match for a close match.	"false"
KEY_PREFER_USER_DICTIONARY		When using AD and UD together, prefer a UD close match over an AD close match	"false"

**USA\_GeocodeConstraints (continued)**

Keys and Constants	Base Equivalent	Description	Default
KEY_EXPANDED_SEARCH		Use an expanded search radius to find a match	"false"
KEY_EXPANDED_SEARCH_RADIUS		The distance to use for the expanded search radius	"25" mi
KEY_EXPANDED_SEARCH_LIMIT_TO_STATE		Do not cross state boundaries when doing an expanded search.	"true"
KEY_CASSRULES		Turn on CASS™ mode. This is an older way of enabling CASS mode, and is still supported in MapMarker Plus 14.0.	"false"
KEY_CASS_DPV		Turn on DPV® mode.	"false"
KEY_ALTERNATE_LOOKUP		Determines whether the preferred lookup is to look for streets first or places first.	STREET_FIRST
KEY_LACSLINK		Turn on LACS <sup>Link</sup> ® mode.	"false"

**Sample Code for Setting Geocoding Constraints**

Following is an example of how to set constraints using the MapMarker USA Java API. This example sets a fallback to Postal Code if a close match could not be made and sets the CASS\_MODE match mode. This uses the `USA_GeocodeConstraints` class.

```
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.setMatchMode(USA_GeocodeConstraints.CASS_MODE);
```

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

```
GeocodeConstraints constraints =new GeocodeConstraints();
constraints.setMustMatchAddressNumber(true);
constraints.setMaxCandidates(1);
constraints.setReturnCloseCandidatesOnly(true);
constraints.setFallbackToPostal(true);
constraints.setMatchMode(GeocodeConstraints.CASS_MODE);
```

### 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.

```
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.

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

Description	Datum	String
Longitude / Latitude	WGS 84	EPSG:4326
Longitude / Latitude	NAD 83	EPSG:4269
Longitude / Latitude	NAD 27 for Continental United States	EPSG:4267

### Geocoding

After 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](#).

```
/* 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.

```
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.

```
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. 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.

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

You can use `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 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 candidate it returns. 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:

```
String resultCode=usaCand.getGeoResult();
```

For more information on interpreting the geocoding results, see the Result Codes chapter in the *MapMarker Desktop User Guide*. This is available at <http://reference.mapinfo.com/>. From that page, select MapInfo MapMarker & MapMarker Plus US then select the most the appropriate version of the MapMarker Plus 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 **Developing Geocoding Applications in Java on page 118**, except use the `MMJClient` class instead of the `MMJEngine` class. The `MMJClient` geocode method requires the URL of the servlet.

```
/* Geocode Address */
String url = "http://mapinfo:8095/mapmarker40/servlet/mapmarker";
MMJClient m_client;
ClientGeocodeResponse geoRes = m_client.geocode(0,inpAddr, geoCon, url);
/* Change the host and port in the url to reflect your host and port */
```

Make sure that your application contains these import statements.

```
/* MapMarker 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\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 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 for Web Client

The following usage example shows 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_UserCandidateAddress;
import com.mapinfo.mapmarker.USA.USA_UserInputAddress;
import com.mapinfo.mapmarker.user.MapMarkerException;
import com.mapinfo.mapmarker.user.MapMarkerFatalException;

public class MMJ_US_Example {

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

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

        USA_GeocodeConstraints geoCon = new USA_GeocodeConstraints();

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

```

```
geoCon.setReturnCloseCandidatesOnly(true);
geoCon.setFallbackToPostal(false);
geoCon.setMatchMode(USA_GeocodeConstraints.CASS_MODE);

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*/

ClientGeocodeResponse geoRes = null;
try
{
    /*Geocode Address */
    geoRes = MMJClient.geocode(MMJClient.GEOCODE_TYPE_ADDRESS,
inpAddr, geoCon, url);
}
catch (MapMarkerException mme)
{
    mme.printStackTrace();
}
catch (MapMarkerFatalException mmfe)
{
    mmfe.printStackTrace();
}
/*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);
    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.user.GeocodeConstraints;
import com.mapinfo.mapmarker.IGeocodeConstraints;
import com.mapinfo.mapmarker.common.AddressImpl;
import com.mapinfo.mapmarker.common.Address;
import com.mapinfo.mapmarker.USA.USA_GeocodeConstraints;
import com.mapinfo.mapmarker.user.MapMarkerJavaAPI;
import com.mapinfo.mapmarker.user.MapMarkerException;
import com.mapinfo.mapmarker.user.MapMarkerFatalException;
import com.mapinfo.mapmarker.CandidateRange;

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);
        constraints.setMatchMode(USA_GeocodeConstraints.CASS_MODE);
        /* ALTERNATIVELY, constraints.setMatchMode("CASSMODE"); */

        Address inpAddr = new AddressImpl();

        /* Set the input Address */
        inpAddr.setCountry("USA");
        inpAddr.setMainAddress(addr); /*street*/
        inpAddr.setAreaName3(city); /*city*/
    }
}

```

```
inpAddr.setAreaName1 (state); /*state*/
inpAddr.setPostCode1(postal); /*zipcode*/

ClientGeocodeResponse geoRes = null;
try
{
    /*Geocode Address */
    geoRes = MMJClient.geocode(MMJClient.GEOCODE_TYPE_ADDRESS,
inpAddr, geoCon, url);
}
catch (MapMarkerException mme)
{
    mme.printStackTrace();
}
catch (MapMarkerFatalException mmfe)
{
    mmfe.printStackTrace();
}

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

/*
 * Get returned candidates
 */
if(candCount > 0)
{
    for (int i=0; i < candCount; i++) {

        /** Get candidate information
         * Review Java docs for a complete list of
         * candidate information
         */

        CandidateAddress candAddr = geoRes.candidateAt(i);

        System.out.println(candAddr);
        System.out.println(candAddr.getLocation());

        System.out.println(
            "Precision Code:" + candAddr.getPrecisionCode());

        System.out.println();

        /** Get candidate range information
         * Review Java docs for a complete list of candidate range
         * information
         */

        for ( int k=0;
            k<candAddr.getNumberOfCandidateRangesFound();k++)
        {
            CandidateRange range = candAddr.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 are using the MapMarker Generic API, use the `AddressImpl` class.

```

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

```

To browse, use the method of the `MMJEngine` class as shown in this example. This class is used for browsing with either the MapMarker USA Java API or the MapMarker Generic Java API. To browse using the `MMJClient` class see [Creating a Web Geocoding Client in Java on page 126](#).

```

/* Browse Address */
MMJEngine engine = new MMJEngine ();
ClientGeocodeResponse geoRes = null;

```

```
geoRes = engine.geocode (MapMarkerJavaAPI.GEOCODE_TYPE_BROWSE, inpAddr,
geoCon);
```

All of the streets that begin with the letter G in the ZIP Code 12180 will be returned. You can use the `USA_UserCandidateRange` 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));

if (candAddr.getNumberOfCandidateRanges () > 0)
    {
        USA_UserCandidateRange candRange = new
USA_UserCandidateRange (candAddr.getRangeAt (0));
        System.out.println ("Candidate Range :" + candRange);
    }
}
}
```

## Geocoding to Geographic Centroids

The MapMarker USA API enables you to geocode records to geographic centroids.

The source data for the city centroids comes from MapInfo's data product *2004 CityInfoAll*, which contains over 220,000 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 centroid, use `MapMarkerJavaAPI.GEOCODE_TYPE_GEOGRAPHIC_CENTROID`.

```
/* Input address example */

USA_UserInputAddress inpAddr = new USA_UserInputAddress ();
inpAddr.setCity ("Albany");
inpAddr.setState ("NY");
```

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

```
AddressImpl inpAddr = new AddressImpl ();
inpAddr.setAreaName3 ("Albany");
inpAddr.setAreaName1 ("NY");
inpAddr.setCountry ("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_GEOGRAPHIC_CENTROID,
inpAddr, geoCon);

```

You can also specify a preference to fallback to geographic centroid if a close match could not be made to a street-level geocode.

```

    /*Set Fallback to Geographic */
    USA_GeocodeConstraints geoCon = new USA_GeocodeConstraints();
    setFallbackToGeographic(true)
    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));
        }
    }

```

Geographic centroid candidates may contain values for areaNames 1-3 (state, county, city), census block, rank, and location. The corresponding georesult codes are: G1 (state), G2 (county), and G3 (city).

For deciding close matches, preference is given to the candidate in which the spelling of the city is closer to the input address. If more than one geographic area with the same name exists (for example, Springfield, VA exists in eight 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 1 to 7, with 1 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, use the `AddressImpl` class.

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

To geocode 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 Generic Java API, 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' */
```

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(USA_CustomGeocoder.GEOCODE_TYPE_AIRPORTS_BY_STATE,
inpAddr, geoCon);
```

All of the known airports located in the specified state will be returned. 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 Generic API, 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' */
```

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(USA_CustomGeocoder.GEOCODE_TYPE_AIRPORTS_BY_CODE,
inpAddr, geoCon);
```

The airport that matches the input code will be returned. 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 (for example, 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, for example, 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 just as it would be without using address point interpolation.

## Determining DPV Availability

The MapMarker USA API enables you to determine DPV<sup>®</sup> usability, status, and expiration.

To obtain detailed information on DPV status (including usability and expiration date) geocode using the following custom geocode type:

```
USA_CustomGeocoder.GEOCODE_TYPE_DPV_STATUS
```

The value is:

```
MapMarkerJavaAPI.GEOCODE_TYPE_CUSTOM_BASE+108
```

## Determining DPV Availability

---

To get the DPV<sup>®</sup> information, geocode using the custom type. The information is returned in the candidate addresses found in the response. The results are returned as strings using the following candidate methods.

Status code	getGenericField1()
Description Message	getGenericField2()
Expiration Date	getGenericField3()

If referencing from a USA user candidate the following methods can be used instead of the genericField# methods.

Status code	getDPVStatusCode()
Description Message	getDPVStatusMsg()
Expiration Date	getDPVExpDate()

The following sample code shows how to use the USA custom geocoding API.

```
/* Get DPV status (presence), status (usability), and expiration */

USA_Constraints usa_prefs = new USA_Constraints();
USA_UserInputAddress usaAddr = new USA_UserInputAddress();
MMJEngine engine = new MMJEngine();
ClientGeocodeResponse resp =
engine.geocode(USA_CustomGeocoder.GEOCODE_TYPE_DPV_STATUS, usaAddr,
usa_prefs);
if (resp.totalPossibleCloseMatchCandidates() > 0)
{
    CandidateAddress cand = resp.candidateAt(0);
    USA_UserCandidateAddress usaCand = new
USA_UserCandidateAddress(cand);
    String statusCodeString = usaCand.getDPVStatusCode();
    String messageString = usaCand.getDPVStatusMsg();
    String expirationDate = usaCand.getDPVExpDate();
}
```

If you are using the MapMarker Generic API, use the AddressImpl class. In this case, the country must be set in the input address.

The following sample code shows generic use of the API (without USA references).

```
Generic use (Not using USA references)

GeocodeConstraints constraints = new GeocodeConstraints ();
Address addr = new AddressImpl();
MMJEngine engine = new MMJEngine();
addr.setCountry("USA");
```

```

ClientGeocodeResponse resp =
engine.geocode(MapMarkerJavaAPI.GEOCODE_TYPE_CUSTOM_BASE+108, addr,
constraints);
if (resp.totalPossibleCloseMatchCandidates() > 0)
{
CandidateAddress cand = resp.candidateAt(0);
String statusCodeString = cand.getGenericField1();
String messageString = cand.getGenericField2();
String expirationDate = cand.getGenericField3();
}

```

The status is returned in CandidateAddress, which contains the following information:

<b>Status Code</b>	<b>Message Returned</b>	<b>DPV Expiration Date (format Month <i>dd</i>, <i>yyyy</i>)</b>
0	DPV <sup>®</sup> is enabled and usable	for example: July 29, 2007
100	No DPV history file present	
101	DPV files not present	
102	DPV is not enabled	
103	DPV data has expired	



# 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** . . . . .142
- ◆ **Using the Sample Application** . . . . .142

## Requirements and Setup

Before you try to geocode an address using MapMarker, 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 can 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 v14 Sample Application.exe) from the \SDK\examples directory where you installed MapMarker. Windows users can also use the Start menu shortcut.

Firm Name :  
Street Name :  
City Name:  
State:  
Postal Code:

**Preferences**

Close Matches Only  
 Fall Back to Postal Code  
 Fall Back to Geographic  
Max Candidates: 3  
Dictionary Options: AD Only  
Geocode Type: Street

**Must Match Options**

House Num  State  City  
 Street  Zipcode  CASS

**MapMarker Server URL**  
http://ruready-W1:8095/mapmarker40/servlet/mapmarker

Server: VALID Test

Close Match	Firm	Street	City	State	Zip	Zip4
-------------	------	--------	------	-------	-----	------

**Candidates Statistics**  
Total Possible Candidates: 0  
Total Returned Candidates: 0  
Total Close Match Candidates: 0

Geocode Address Close

## Testing the Server

The server must be running properly before you can geocode. To test the server, use the sample application's URL server test area. The URL to the MapMarker 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 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 USA)

## 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.

# Using the XML API in .NET Framework

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.

We provide a sample XML client application. This sample application is coded in C# and built as a Visual Studio .NET project, and is located under the resources of media.

The sample application allows 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.

To open the project, you must use Visual Studio .NET: It is assumed that the developer can connect to a running MapMarker Server.

## In this chapter:

- ◆ **MapMarker XML Programming in .NET Framework . . . . .146**
- ◆ **Interacting with a MapMarker Server . . . . .146**
- ◆ **Using XSD Files to Create .NET Classes . . . . .147**
- ◆ **Tips for Adding Code for Your Application . . . . .148**
- ◆ **Examples of XML Requests and Responses . . . . .151**
- ◆ **Performance Tips . . . . .161**
- ◆ **Summary of the MapMarker XML API. . . . .161**

## 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 attempts to carry out the requested operation, and if successful, sends an XML response message, which conforms to the MapMarker response schema.

The MapMarker schemas are installed by MapMarker in:

MapMarker\_CNT\_v4\sdk\engine\schema, where *CNT* is the country identifier. The schemas are:

- MMJRequest.xsd
- MMJResponse.xsd
- MMJCommon.xsd

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 then must 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 must also 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.  
For more detailed examples, see [Examples of XML Requests and Responses on page 151](#).
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 contains all of the required information to generate an XML message or parse the results of a response. This method requires a significant 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 provides an overview of this approach as well as provide some practical examples. See:

- [Using XSD Files to Create .NET Classes, p. 147](#)
- [Tips for Adding Code for Your Application, p. 148](#)
- [Examples of XML Requests and Responses, p. 151](#)
- [Performance Tips, p. 161](#)
- [Summary of the MapMarker XML API, p. 161](#)

## 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:

- 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.

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

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. All of these schemas are installed in your product `\sdk\engine\schema` folder. The XSD tool puts these common classes in both of the class files that it creates which causes 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.

## Tips for Adding Code for Your Application

This section describes some guidelines and shows examples of coding your application.

**Declare geocoding objects** The following example shows how to declare geocoding objects.

```
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;
```

**Set the input address properties.** The following example shows how to set the properties for the input address object. This example illustrates street address geocoding for U.S addresses. Different countries have different input address requirements.

```
inputAddress.Address.MainAddress = "1 Global View";
inputAddress.Address.AreaName3="Troy";
inputAddress.Address.AreaName1="NY";
inputAddress.Address.postCode1="12180";
inputAddress.Address.Country="USA";
```

**Instantiate request constraints and add base constraints.** The MapMarker server supports a number of different geocoding constraints. These are classified as base constraints, additional constraints, and country-specific constraints.

The following example shows how to set the base geocoding constraints by setting the constraint name and constraint values.

```
geocodeRequest.GeocodeConstraints= new GeocodeConstraintsType()
geocodeRequest.GeocodeConstraints.BaseConstraints = new
BaseConstraintsType();

/* Setting Base Constraints */
geocodeRequest.GeocodeConstraints.BaseConstraints.closeMatchesOnly =
true;
geocodeRequest.GeocodeConstraints.BaseConstraints.closeMatchesOnlySpecifi
ed = true;

geocodeRequest.GeocodeConstraints.BaseConstraints.fallbackToGeographicCen
troid= true;
```

```
geocodeRequest.GeocodeConstraints.BaseConstraints.fallbackToGeographicCentroidSpecified = true;
```

**Note** For a complete list of the base and additional geocoding constraints, see [Geocoding Constraints on page 164](#).

**Add additional geocoding constraints.** The following examples shows how to set the additional geocoding constraints by using the constraint name and constraint values. The first example sets the MustMatch conditions for Address Number and Main Address. These conditions must be met in order to return a close match.

```
/* Setting additional Constraints */
geocodeRequest.GeocodeConstraints.AddConstraint("MustMatchAddrNum",
"true");
geocodeRequest.GeocodeConstraints.AddConstraint("MustMatchMainAddr",
"true");
```

The following sample code shows how to define Exact mode matching. This uses the MatchMode additional constraint with the ExactMode value. Base and additional constraints are listed in the table of [Geocoding Constraints](#) beginning on [page 164](#). The MatchMode constraints are described on [page 166](#).

```
<GeocodeConstraints>
  <BaseConstraints></BaseConstraints>
  <AdditionalConstraints>
    <KeyValuePair>
      <Key>MatchMode</Key>
      <Value>ExactMode</Value>
    </KeyValuePair>
  </AdditionalConstraints>
</GeocodeConstraints>
```

For a complete list of the additional geocoding constraints, see the table [Geocoding Constraints on page 164](#). Additional constraints are indicated in Constraint Name column of the table.

**Add CASS Mode constraints.** The following sample code shows how to enable CASS mode using MatchMode additional constraint.

```
<GeocodeConstraints>
  <BaseConstraints></BaseConstraints>
  <AdditionalConstraints>
    <KeyValuePair>
      <Key>MustMatchAddrNum</Key>
      <Value>>true</Value>
    </KeyValuePair>
    <KeyValuePair>
      <Key>KEY_EXPANDED_SEARCH_RADIUS</Key>
      <Value>25</Value>
    </KeyValuePair>
    <KeyValuePair>
      <Key>MatchMode</Key>
      <Value>CASSMODE</Value>
    </KeyValuePair>
    <KeyValuePair>
      <Key>KEY_USE_EXPANDED_SEARCH</Key>
```

```
<Value>>false</Value>
</KeyValuePair>
<KeyValuePair>
  <Key>KEY_EXPANDED_SEARCH_LIMIT_TO_STATE</Key>
  <Value>>true</Value>
</KeyValuePair>
<KeyValuePair>
  <Key>KEY_ALTERNATE_LOOKUP</Key>
  <Value>STREET_FIRST</Value>
</KeyValuePair>
</AdditionalConstraints>
</GeocodeConstraints>
```

For a complete list of the country-specific geocoding constraints, see the table [Geocoding Constraints on page 164](#). Country-specific constraints are indicated in the Constraint Name column of the table.

**Specify the geocoding request type.** The MapMarker server supports a number of different types of geocoding and browsing requests.

See the entry REQUEST TYPE VALUES in the GEOCODING CONSTRAINTS table for a list of the different request types.

**Set the appropriate HTTP headers.** HTTP headers are associated with each MapMarker request. The following example code shows 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_3_0");
web.Headers.Add("MI_XMLProtocolTransactionId", "0000");
```

**Set the additional fields.** This example illustrates setting the RESULT\_CODE additional field. For additional field values see the table [Candidate Additional Field Values on page 169](#).

```
{
  if (addFields[i].Key == "RESULT_CODE")
    georesult = addFields[i].Value;
}

this.lblOutputAddress.Text = mainAddress+", "+city+", "+province+",
"+postcode;
this.lblAddOutput.Text = georesult+" ["+latitude+", "+longitude+"]";
```

## 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

```
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

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

### 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");
web.Headers.Add("MI_XMLProtocolRequest", "DictionarySearchOrderRequest");
```

### Get Constraints Request

```
web.Headers.Clear();
web.Headers.Add("Content-Type", "text/xml");
web.Headers.Add("MI_XMLProtocolRequest", "ConstraintsRequest");
```

## Examples of XML Requests and Responses

This section includes some short snippets of sample code that illustrate requests and responses using the MapMarker XML API.

### Street Request and Response

This example submits a street level geocoding request.

```
<?xml version="1.0" encoding="UTF-8"?>
<RequestEnvelope>
  <GeocodeRequest requestID="9D0711B1-8175-4AAE-A76B-E1B18A396626">
    <InputAddress addressID="D4C189FF-E987-C47D-48BD-C2F64C030056">
      <Address>
        <AreaName1>NY</AreaName1>
        <AreaName3>Troy</AreaName3>
        <Country>USA</Country>
        <MainAddress>1 Global vw</MainAddress>
        <postCode1>12180</postCode1>
      </Address>
    </InputAddress>
    <GeocodeConstraints>
```

```
<BaseConstraints dictionaryUsage="AD_ONLY" />
<AdditionalConstraints>
  <KeyValuePair>
    <Key>MustMatchAddrNum</Key>
    <Value>true</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>KEY_EXPANDED_SEARCH_RADIUS</Key>
    <Value>25</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>MatchMode</Key>
    <Value>CASSMODE</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>KEY_USE_EXPANDED_SEARCH</Key>
    <Value>>false</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>KEY_EXPANDED_SEARCH_LIMIT_TO_STATE</Key>
    <Value>true</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>KEY_ALTERNATE_LOOKUP</Key>
    <Value>STREET_FIRST</Value>
  </KeyValuePair>
</AdditionalConstraints>
</GeocodeConstraints>
</GeocodeRequest>
</RequestEnvelope>
```

The following example shows the response from the above street level geocoding request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ResponseEnvelope>
  <GeocodeResponse requestID="9D0711B1-8175-4AAE-A76B-E1B18A396626">
    <ResponseCode>0</ResponseCode>
    <RequestResult addressID="D4C189FF-E987-C47D-48BD-C2F64C030056">
      <ResponseCode>0</ResponseCode>
      <GeocodeSummary>
        <TotalLocationsFound>4</TotalLocationsFound>
        <TotalLocationsReturned>3</TotalLocationsReturned>
        <TotalCloseMatchesFound>1</TotalCloseMatchesFound>
        <UniqueCloseMatchFound>true</UniqueCloseMatchFound>
        <dataLicensed>true</dataLicensed>
      </GeocodeSummary>
      <Candidate genericField4Matched="false" areaName3Matched="true"
streetNameMatched="true" postCode2Matched="true"
genericField2Matched="false" closeMatch="true"
fmtdAddr="1 GLOBAL VW" srcStID="18344492" intersection="false"
addressNumberMatched="true" postCode1Matched="true"
preDirectionalMatched="true" CfgOrder="1" fromUserDictionary="false"
areaName1Matched="true" placeNameMatched="true"
streetPrefixSuffixMatched="true" areaName2Matched="true"
```

```

postDirectionalMatched="true" genericField3Matched="false"
streetNameFieldsMatched="true" areaName4Matched="true"
thoroughfareTypeMatched="true" matchPrecision="1"
countryMatched="true" genericField1Matched="false"
fmtdLoc="TROY, NY 12180-8371">
  <Address>
    <AdditionalFields>
      <KeyValuePair>
        <Key>RESULT_CODE</Key>
        <Value>S5HPNTSCZA</Value>
      </KeyValuePair>
      <KeyValuePair>
        <Key>DELIVERY_POINT</Key>
        <Value>01</Value>
      </KeyValuePair>
      <KeyValuePair>
        <Key>REC_TYPE</Key>
        <Value>S</Value>
      </KeyValuePair>
      <KeyValuePair>
        <Key>CENSUS_BLOCK</Key>
        <Value>360830523018010</Value>
      </KeyValuePair>
      <KeyValuePair>
        <Key>CARRIER_ROUTE</Key>
        <Value>C034</Value>
      </KeyValuePair>
      <KeyValuePair>
        <Key>CHECK_DIGIT</Key>
        <Value>8</Value>
      </KeyValuePair>
    </AdditionalFields>
    <AddressNumber>1</AddressNumber>
    <AreaName1>NY</AreaName1>
    <AreaName3>TROY</AreaName3>
    <Country>USA</Country>
    <MainAddress>GLOBAL</MainAddress>
    <postCode1>12180</postCode1>
    <postCode2>8371</postCode2>
    <postThoroughfareType>VW</postThoroughfareType>
  </Address>
  <Point srsName="epsg:4326">
    <coord>
      <X>-73.702967</X>
      <Y>42.682874</Y>
    </coord>
  </Point>
  <TotalRangesFound>0</TotalRangesFound>
</Candidate>
<Candidate genericField4Matched="false" areaName3Matched="true"
streetNameMatched="true" postCode2Matched="true"
genericField2Matched="false" closeMatch="false" fmtdAddr="GLOBAL VW"
srcStID="18344492" intersection="false" addressNumberMatched="true"

```

```
postCode1Matched="true" preDirectionalMatched="true" CfgOrder="1"
fromUserDictionary="false" areaName1Matched="true"
placeNameMatched="true" streetPrefixSuffixMatched="true"
areaName2Matched="true" postDirectionalMatched="true"
genericField3Matched="false" streetNameFieldsMatched="true"
areaName4Matched="true" thoroughfareTypeMatched="true"
matchPrecision="2" countryMatched="true"
genericField1Matched="false" fmtLoc="TROY, NY 12180">
  <Address>
    <AdditionalFields>
      <KeyValuePair>
        <Key>RESULT_CODE</Key>
        <Value>S4HPNTSCZA</Value>
      </KeyValuePair>
      <KeyValuePair>
        <Key>CENSUS_BLOCK</Key>
        <Value>360830523018010</Value>
      </KeyValuePair>
    </AdditionalFields>
    <AreaName1>NY</AreaName1>
    <AreaName3>TROY</AreaName3>
    <Country>USA</Country>
    <MainAddress>GLOBAL</MainAddress>
    <postCode1>12180</postCode1>
    <postThoroughfareType>VW</postThoroughfareType>
  </Address>
  <Point srsName="epsg:4326">
    <coord>
      <X>-73.702626</X>
      <Y>42.682999</Y>
    </coord>
  </Point>
  <TotalRangesFound>0</TotalRangesFound>
</Candidate>
<Candidate genericField4Matched="false" areaName3Matched="true"
streetNameMatched="true" postCode2Matched="true"
genericField2Matched="false" closeMatch="false"
fmtdAddr="3 GLOBAL VW" srcStID="18344492" intersection="false"
addressNumberMatched="false" postCode1Matched="true"
preDirectionalMatched="true" CfgOrder="1" fromUserDictionary="false"
areaName1Matched="true" placeNameMatched="true"
streetPrefixSuffixMatched="true" areaName2Matched="true"
postDirectionalMatched="true" genericField3Matched="false"
streetNameFieldsMatched="true" areaName4Matched="true"
thoroughfareTypeMatched="true" matchPrecision="1"
countryMatched="true" genericField1Matched="false"
fmtLoc="TROY, NY 12180-8371">
  <Address>
    <AdditionalFields>
      <KeyValuePair>
        <Key>RESULT_CODE</Key>
        <Value>S5-PNTSCZA</Value>
      </KeyValuePair>
```

```

    <KeyValuePair>
      <Key>DELIVERY_POINT</Key>
      <Value>03</Value>
    </KeyValuePair>
    <KeyValuePair>
      <Key>CENSUS_BLOCK</Key>
      <Value>360830523018010</Value>
    </KeyValuePair>
    <KeyValuePair>
      <Key>CHECK_DIGIT</Key>
      <Value>6</Value>
    </KeyValuePair>
  </AdditionalFields>
  <AddressNumber>3</AddressNumber>
  <AreaName1>NY</AreaName1>
  <AreaName3>TROY</AreaName3>
  <Country>USA</Country>
  <MainAddress>GLOBAL</MainAddress>
  <postCode1>12180</postCode1>
  <postCode2>8371</postCode2>
  <postThoroughfareType>VW</postThoroughfareType>
</Address>
<Point srsName="epsg:4326">
  <coord>
    <X>-73.702356</X>
    <Y>42.683019</Y>
  </coord>
</Point>
  <TotalRangesFound>0</TotalRangesFound>
</Candidate>
</RequestResult>
</GeocodeResponse>
</ResponseEnvelope>

```

## Postal Request and Response

This example shows a postal geocoding request:

```

<?xml version="1.0" encoding="UTF-8"?>
<RequestEnvelope>
  <GeocodeRequest requestID="E7CBF8CE-27DF-61BC-6D8D-75CE1E2035A9">
    <InputAddress addressID="32A0DB1B-DB1D-BDAE-D2EA-A198354E689D">
      <Address>
        <AreaName1>NY</AreaName1>
        <AreaName3>Troy</AreaName3>
        <Country>USA</Country>
        <MainAddress>1 Global vw</MainAddress>
        <postCode1>12180</postCode1>
      </Address>
    </InputAddress>
    <GeocodeConstraints>
      <BaseConstraints dictionaryUsage="AD_ONLY" requestType="1" />
    </GeocodeConstraints>
  </GeocodeRequest>
</RequestEnvelope>

```

```
<AdditionalConstraints>
  <KeyValuePair>
    <Key>MustMatchAddrNum</Key>
    <Value>>true</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>KEY_EXPANDED_SEARCH_RADIUS</Key>
    <Value>25</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>KEY_USE_EXPANDED_SEARCH</Key>
    <Value>>false</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>KEY_EXPANDED_SEARCH_LIMIT_TO_STATE</Key>
    <Value>>true</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>KEY_ALTERNATE_LOOKUP</Key>
    <Value>STREET_FIRST</Value>
  </KeyValuePair>
</AdditionalConstraints>
</GeocodeConstraints>
</GeocodeRequest>
</RequestEnvelope>
```

The following example shows the response from the above postal geocoding request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ResponseEnvelope>
  <GeocodeResponse requestID="20060427125326648">
    <ResponseCode>0</ResponseCode>
    <RequestResult addressID="12345">
      <ResponseCode>0</ResponseCode>
      <GeocodeSummary>
        <TotalLocationsFound>1</TotalLocationsFound>
        <TotalLocationsReturned>1</TotalLocationsReturned>
        <TotalCloseMatchesFound>1</TotalCloseMatchesFound>
        <UniqueCloseMatchFound>>true</UniqueCloseMatchFound>
        <dataLicensed>>true</dataLicensed>
      </GeocodeSummary>
      <Candidate genericField4Matched="false" areaName3Matched="true"
streetNameMatched="false" postCode2Matched="true"
genericField2Matched="false" closeMatch="true" fmtdAddr=""
srcStID="0" intersection="false" addressNumberMatched="false"
postCode1Matched="true" preDirectionalMatched="false" CfgOrder="1"
fromUserDictionary="false" areaName1Matched="true"
placeNameMatched="false" streetPrefixSuffixMatched="false"
areaName2Matched="false" postDirectionalMatched="false"
genericField3Matched="false" streetNameFieldsMatched="false"
areaName4Matched="false" thoroughfareTypeMatched="false"
matchPrecision="3" countryMatched="true"
genericField1Matched="false" fmtdLoc="">
        <Address>
```

```

    <AdditionalFields>
      <KeyValuePair>
        <Key>RESULT_CODE</Key>
        <Value>Z1</Value>
      </KeyValuePair>
      <KeyValuePair>
        <Key>CENSUS_BLOCK</Key>
        <Value>36083</Value>
      </KeyValuePair>
    </AdditionalFields>
    <AreaName1>NY</AreaName1>
    <AreaName3>TROY</AreaName3>
    <Country>USA</Country>
    <postCode1>12180</postCode1>
  </Address>
  <Point srsName="epsg:4326">
    <coord>
      <X>-73.65733</X>
      <Y>42.72989</Y>
    </coord>
  </Point>
  <TotalRangesFound>0</TotalRangesFound>
</Candidate>
</RequestResult>
</GeocodeResponse>
</ResponseEnvelope>

```

## Highway Exit Request and Response

This example submits a highway exit geocoding request:

```

<?xml version="1.0" encoding="UTF-8"?>
<RequestEnvelope clientLocale="en_US" encoding="UTF-8">
  <GeocodeRequest requestID="20060427124917601">
    <InputAddress addressID="12345">
      <Address>
        <AreaName1>NY</AreaName1>
        <AreaName3 />
        <AreaName4 />
        <Country>USA</Country>
        <GenericField1 />
        <MainAddress>I 90 Exit 8</MainAddress>
        <placeName />
        <postCode1 />
      </Address>
      <GeocodeConstraints>
        <BaseConstraints clientLocale="en_US"
          closeMatchesOnly="false"
          fallbackToPostalCentroid="false"
          fallbackToGeographicCentroid="false"
          dictionaryUsage="PREFER_AD" requestType="201">
          <maxCandidates>3</maxCandidates>
        </BaseConstraints>
      </GeocodeConstraints>
    </InputAddress>
  </GeocodeRequest>
</RequestEnvelope>

```

```
<maxRanges>0</maxRanges>
<offsetFromCorner unitsOfMeasure="m">10</offsetFromCorner>
<offsetFromStreet unitsOfMeasure="m">10</offsetFromStreet>
</BaseConstraints>
<AdditionalConstraints>
  <KeyValuePair>
    <Key>MustMatchAddrNum</Key>
    <Value>>false</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>MustMatchMainAddr</Key>
    <Value>>false</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>MustMatchArea3</Key>
    <Value>>false</Value>
  </KeyValuePair>
  <KeyValuePair>
    <Key>MustMatchPostalCode</Key>
    <Value>>false</Value>
  </KeyValuePair>
</AdditionalConstraints>
</GeocodeConstraints>
</InputAddress>
</GeocodeRequest>
</RequestEnvelope>
```

The following example shows the response from the above highway exit geocoding request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ResponseEnvelope>
  <GeocodeResponse requestID="20060427124917601">
    <ResponseCode>0</ResponseCode>
    <RequestResult addressID="12345">
      <ResponseCode>0</ResponseCode>
      <GeocodeSummary>
        <TotalLocationsFound>4</TotalLocationsFound>
        <TotalLocationsReturned>3</TotalLocationsReturned>
        <TotalCloseMatchesFound>0</TotalCloseMatchesFound>
        <UniqueCloseMatchFound>>false</UniqueCloseMatchFound>
        <dataLicensed>>true</dataLicensed>
      </GeocodeSummary>
      <Candidate genericField4Matched="false" areaName3Matched="false"
streetNameMatched="false" postCode2Matched="false"
genericField2Matched="false" closeMatch="false" fmtdAddr="90"
srcStID="0" intersection="false" addressNumberMatched="false"
postCode1Matched="false" preDirectionalMatched="false" CfgOrder="1"
fromUserDictionary="false" areaName1Matched="false"
placeNameMatched="false" streetPrefixSuffixMatched="false"
areaName2Matched="false"
postDirectionalMatched="false" genericField3Matched="false"
streetNameFieldsMatched="false" areaName4Matched="false"
thoroughfareTypeMatched="false" matchPrecision="7"
```

```

countryMatched="false" genericField1Matched="false" fmtdLoc="">
  <Address>
    <AreaName1>NY</AreaName1>
    <AreaName3>RENSSELAER</AreaName3>
    <GenericField1>0004233831</GenericField1>
    <GenericField2>8</GenericField2>
    <GenericField3>STATE HWY 43</GenericField3>
    <MainAddress>90</MainAddress>
    <postCode1>12144</postCode1>
    <preThoroughfareType>I</preThoroughfareType>
  </Address>
  <Point srsName="epsg:4326">
    <coord>
      <X>-73.711308</X>
      <Y>42.647364</Y>
    </coord>
  </Point>
  <TotalRangesFound>0</TotalRangesFound>
</Candidate>
  <Candidate genericField4Matched="false" areaName3Matched="false"
streetNameMatched="false" postCode2Matched="false"
genericField2Matched="false" closeMatch="false" fmtdAddr="90" srcStID="0"
intersection="false" addressNumberMatched="false"
postCode1Matched="false" preDirectionalMatched="false" CfgOrder="1"
fromUserDictionary="false" areaName1Matched="false"
placeNameMatched="false" streetPrefixSuffixMatched="false"
areaName2Matched="false" postDirectionalMatched="false"
genericField3Matched="false" streetNameFieldsMatched="false"
areaName4Matched="false" thoroughfareTypeMatched="false"
matchPrecision="7" countryMatched="false" genericField1Matched="false"
fmtdLoc="">
  <Address>
    <AreaName1>NY</AreaName1>
    <AreaName3>RENSSELAER</AreaName3>
    <GenericField1>0004233832</GenericField1>
    <GenericField2>8</GenericField2>
    <MainAddress>90</MainAddress>
    <postCode1>12144</postCode1>
    <preThoroughfareType>I</preThoroughfareType>
  </Address>
  <Point srsName="epsg:4326">
    <coord>
      <X>-73.711308</X>
      <Y>42.647364</Y>
    </coord>
  </Point>
  <TotalRangesFound>0</TotalRangesFound>
</Candidate>
  <Candidate genericField4Matched="false" areaName3Matched="false"
streetNameMatched="false" postCode2Matched="false"
genericField2Matched="false" closeMatch="false" fmtdAddr="90" srcStID="0"
intersection="false" addressNumberMatched="false"
postCode1Matched="false" preDirectionalMatched="false" CfgOrder="1"

```

```
fromUserDictionary="false" areaName1Matched="false"
placeNameMatched="false" streetPrefixSuffixMatched="false"
areaName2Matched="false" postDirectionalMatched="false"
genericField3Matched="false" streetNameFieldsMatched="false"
areaName4Matched="false" thoroughfareTypeMatched="false"
matchPrecision="7" countryMatched="false" genericField1Matched="false"
fmtLoc="">
  <Address>
    <AreaName1>NY</AreaName1>
    <AreaName3>RENSELAER</AreaName3>
    <GenericField1>0004233951</GenericField1>
    <GenericField2>8</GenericField2>
    <GenericField3>STATE HWY 43</GenericField3>
    <MainAddress>90</MainAddress>
    <postCode1>12144</postCode1>
    <preThoroughfareType>I</preThoroughfareType>
  </Address>
  <Point srsName="epsg:4326">
    <coord>
      <X>-73.718504</X>
      <Y>42.655088</Y>
    </coord>
  </Point>
  <TotalRangesFound>0</TotalRangesFound>
</Candidate>
  <Candidate genericField4Matched="false" areaName3Matched="false"
streetNameMatched="false" postCode2Matched="false"
genericField2Matched="false" closeMatch="false" fmtAddr="90" srcStID="0"
intersection="false" addressNumberMatched="false"
postCode1Matched="false" preDirectionalMatched="false" CfgOrder="1"
fromUserDictionary="false" areaName1Matched="false"
placeNameMatched="false" streetPrefixSuffixMatched="false"
areaName2Matched="false" postDirectionalMatched="false"
genericField3Matched="false" streetNameFieldsMatched="false"
areaName4Matched="false" thoroughfareTypeMatched="false"
matchPrecision="7" countryMatched="false" genericField1Matched="false"
fmtLoc="">
  <Address>
    <AreaName1>NY</AreaName1>
    <AreaName3>RENSELAER</AreaName3>
    <GenericField1>0004233952</GenericField1>
    <GenericField2>8</GenericField2>
    <MainAddress>90</MainAddress>
    <postCode1>12144</postCode1>
    <preThoroughfareType>I</preThoroughfareType>
  </Address>
  <Point srsName="epsg:4326">
    <coord>
      <X>-73.718504</X>
      <Y>42.655088</Y>
    </coord>
  </Point>
  <TotalRangesFound>0</TotalRangesFound>
```

```

    </Candidate>
  </RequestResult>
</GeocodeResponse>
</ResponseEnvelope>

```

## Setting the Coordinate System

The coordinate system is set as part of the base constraints of the request. The default coordinate system is WGS 84 (EPSG: 4326). The following code sample shows how to change the client coordinate system to NAD 83 (EPSG: 4269). The code and codeSpace values for the CoordinateReferenceSystemTypeIdentifier include EPSG SRS Names and MapInfo MapBasic Projection String SRS Names.

```

GeocodeConstraintsType gc = new GeocodeConstraintsType();
geoRequest.GeocodeConstraints = gc;
geoRequest.GeocodeConstraints.BaseConstraints = new
BaseConstraintsType();
CoordinateReferenceSystemType csys = new CoordinateReferenceSystemType();
CoordinateReferenceSystemTypeIdentifier ident = new
CoordinateReferenceSystemTypeIdentifier();
ident.code = "4269";
ident.codeSpace = "EPSG";
csys.Identifier = ident;
gc.BaseConstraints.CoordinateReferenceSystem = csys;

```

## Performance Tips

**Batch Performance Improvement** Increasing the number of addresses in a batch file will improve performance. For example, submitting a batch file with from 25 to 50 addresses may increase geocoding speed by 50 percent compared to submitting multiple batch files, with each file containing a single address. The optimal batch size varies depending on type of table, server hardware. or network conditions. In general, performance gains will level off between 25 and 35 addresses per file.

Performance may decrease (or cause client/server out of memory errors) when the batch file gets extremely large.

Setting max candidates = 1 in the XML batch request will also improve client/server performance.

**Note** Only the XML API supports batch request files. This feature is not available with the Java API

## Summary of the MapMarker XML API

This section summarizes the MapMarker XML API classes, constraints, and geocoding request types.

## CandidateAddress Class

`com.mapinfo.mapmarker.CandidateAddress`

The CandidateAddress class represents the CandidateAddress that is returned from call to a GeocodableAddress. The CandidateAddress contains CandidateRanges if range data was found for the address. The Values in this table represent precision codes.

**CandidateAddress Class**

Key	Value	Description
ADDRESS_POINT_INTERPOLATED	17	Precision code indication that the result was generated by using address point data to modify the candidates segment data.
ADDRESS_POINT_PRECISION	16	precision code indicating the result is an Address Point.
GEOGRAPHIC_AREANAME1	8	Match level indicating that the candidate point represents an area name 1 centroid for this candidate address.
GEOGRAPHIC_AREANAME2	9	Match level indicating that the candidate point represents an area name 2 centroid for this candidate address.
GEOGRAPHIC_AREANAME3	10	Match level indicating that the candidate point represents an area name 3 centroid for this candidate address.
GEOGRAPHIC_AREANAME4	11	Match level indicating that the candidate point represents an area name 4 centroid for this candidate address.
LOCAL_CUSTOM_POINT_PRECISION_1	12	Additional point precision for unspecified custom item.
LOCAL_CUSTOM_POINT_PRECISION_2	13	Additional point precision for unspecified custom item.
LOCAL_CUSTOM_POINT_PRECISION_3	14	Additional point precision for unspecified custom item
LOCAL_CUSTOM_POINT_PRECISION_4	15	Additional point precision for unspecified custom item,
NO_COORDINATES_AVAILABLE	0	Match level indicating that no coordinate information is available for this candidate address.

**CandidateAddress Class (continued)**

Key	Value	Description
POINT_OF_INTEREST	7	Match level indicating that the candidate point represents a point of interest for this candidate address.
POSTAL_LEVEL_POSTAL_CODE_1	13	Match level indicating that the candidate point represents postal code 1 centroid for this candidate address.
POSTAL_LEVEL_POSTAL_CODE_2	5	Match level indicating that the candidate point represents a centroid of a postal code 2 for this candidate address.
POSTAL_LEVEL_POSTAL_CODE_2_PARTIAL	4	Match level indicating that the candidate point represents a centroid of a partial postal code 2 for this candidate address.
STREET_LEVEL_INTERPOLATED	1	Match level indicating that the candidate point represents an interpolated value for this candidate address.
STREET_LEVEL_INTERSECTION	6	Match level indicating that the candidate point represents an intersection for this candidate address.
STREET_LEVEL_SHAPE_PATH	2	Match level indicating that the candidate point represents a street segment midpoint for this candidate address.

**CandidateRange Class**

`com.mapinfo.mapmarker.CandidateRange`

The CandidateRange class contains information about a candidate's ranges.

**CandidateRange Class**

Key	Value	Description
ODD_EVEN_BOTH	0	Range contains both odd and even house numbers.
ODD_EVEN_EVEN	2	Range contains even house numbers.
ODD_EVEN_ODD	1	Range contains odd house numbers.

**CandidateRange Class (continued)**

Key	Value	Description
ODD_EVEN_UNKNOWN	-1	Unknown house number odd/even status for this range.
STREET_SIDE_LEFT	1	Range is on the left side of the street.
STREET_SIDE_RIGHT	2	Range is on the right side of the street.
STREET_SIDE_UNKNOWN	0	Unknown street side information for this range.

**Geocoding Constraints**

`com.mapinfo.mapmarker.user.GeocodeConstraints`

The GeocodeConstraints class provides a generic implementation for setting geocoding engine properties. It defines constants that are used by local country geocoders for standard geocoding settings.

The following tables describes the base constraints, additional constraints, and country-specific constraints that you can use in your application. The type of constraint (Base, Additional, or Country-specific) is indicated in Constraint Name column of the table. This table also includes the **Request Types Values**

**Geocoding Constraints**

Constraint Name (Base, Additional, Country)	Value	Description
AddrPntInterp Base	true>false	Address Point Interpolation
ClientLocale Base	locale string	Used to set different locales.
CloseMatchesOnly Base	true>false	Sets Close Matches Only.

## Geocoding Constraints (continued)

Constraint Name (Base, Additional, Country)	Value	Description
CoordinateReferenceSystem Base	CoordinateReference SystemType	String describing the client coordinate system. Accepted values include EPSG SRS Names as well as MAPINFO MAPBASIC Projection String SRS Names. Default uses WGS 84
CoordinateReferenceSystemType Base	CoordinateReference SystemTypeIdentifier	Holds the CoordinateReferenceSystem TypeIdentifier
CoordinateReferenceSystemType Identifier Base	Code (string) CodeSpace (string)	example:4326 example: EPSG
DictionarySearchOrderListType Base	CfgOrder = number	AllowSearch = true/false Description = string UserPriority = number IsUserDictionary = true/false
DictionarySearchOrderResponse Base	DictionarySearch OrderResponseType	DictionarySearch OrderListCount = number DictionarySearch OrderList
DictionaryUsage Base	DictionaryUsageType	Sets the dictionary usage preference (Address, User, or both).  Enumerator {AD_ONLY, UD_ONLY, AD_AND_UD, PREFER_AD, PREFER_UD}
FallbackToGeographicCentroid Base	true/false	Sets fallback to geographic centroid.
FallbackToPostalCentroid Base	true/false	Sets fallback to postal centroid.

**Geocoding Constraints (continued)**

<b>Constraint Name (Base, Additional, Country)</b>	<b>Value</b>	<b>Description</b>
MatchMode Additional	RelaxedMode ExactMode DefaultMode CASSMODE (U.S. ONLY)  See <b>Candidate Additional Field Values on page 169.</b>	Sets the MatchMode. These match modes provide defined levels of matching precision.
MaxCandidates Base	0 – 99, -1 returns all	Sets the maximum number of returned candidates.
MaxRanges Base	0 – 99, -1 returns all	Sets the maximum number of ranges that is returned for each candidate
MaxRangeUnits Base	0 – 99, -1 returns all	Maximum units per range to return.
MustMatchAddrNum Additional	true>false	Only candidates that match the main address number are considered a close match.
MustMatchArea1 Additional	true>false	Only candidates that match AreaName1 are considered a close match.
MustMatchArea2 Additional	true>false	Only candidates that match AreaName2 are considered a close match.
MustMatchArea3 Additional	true>false	Only candidates that match AreaName3 are considered a close match.
MustMatchArea4 Additional	true>false	Only candidates that match AreaName4 are considered a close match.
MustMatchCountry Additional	true>false	Only candidates that match the country are considered a close match.

## Geocoding Constraints (continued)

Constraint Name (Base, Additional, Country)	Value	Description
MustMatchInput Additional	true/false	Only candidates that match all user input are considered a close match.
MustMatchMainAddr Additional	true/false	Only candidates that match the main address are considered a close match
MustMatchPostalCode Additional	true/false	Only candidates that match the postal code are considered a close match.
OffsetFromCorner Base	OffsetType	Sets the offset distance from the corner option.
OffsetFromStreet Base	OffsetType	Sets the position of the geocoded point with respect to the centerline of the street.
OffsetType Base	Unit of measure = {ft,m,mi,km} Value = 0 - 100	Sets the key for the units on the corner offset option
KEY_ALTERNATE_LOOKUP	true/false	Determines whether the preferred lookup is or streets first or places first.
KEY_CASS_DPV	true/false	Turn on DPV <sup>®</sup> mode.
KEY_CASS_RULES	true/false	Turn on CASS <sup>™</sup> mode.
KEY_EXPANDED_SEARCH_LIMIT_TO_STATE	true/false	Do not cross state boundaries when doing an expanded search.
KEY_EXPANDED_SEARCH_RADIUS	0-99 miles	The distance to use for the expanded search radius.
KEY_USE_EXPANDED_SEARCH	true/false	Use an expanded search radius to find a match
LACSLINK	true/false	Turn on LACS <sup>Link</sup> <sup>®</sup> mode.
ZIPOVERCITY	true/false	Candidates matching on postal code are preferred over city matches.

**Geocoding Constraints (continued)**

<b>Constraint Name (Base, Additional, Country)</b>	<b>Value</b>	<b>Description</b>
PLACE_FIRST	true\false	Candidates matching on place are preferred over street.
STREET_FIRST	true\false	Candidates matching on street are preferred over place.
RequestType	0-500	Type of geocoding request. See <b>Request Types Values</b> in this table.
<b>Request Types Values Specifying the type of geocoding request</b>		These are used as values for <b>RequestType</b> constraints.
Geocode Address	0	
Geocode Postal Centroid	1	
Geocode Geographic Centroid	4	
Browse	6	
Geocode Highway Exits	201	
Geocode Airport by Code	202	
Geocode Airport by State	203	
Geocode County FIPS List	204	
DB Availability	205	
DPV Support	206	
LACS LINK Supported	207	
DPV Status	208	

## Candidate Additional Field Values

The following values are used with AdditionalFields <KeyValuePair> statements.

**Candidate Additional Field Values**

Candidate Additional Field Values	Example	Country
RESULT_CODE	S5HPNTSCZA	All Countries
DELIVERY_POINT	01	USA
CENSUS_BLOCK	360830523018010	USA
CARRIER_ROUTE	C034	USA
LACS	L	USA
CHECK_DIGIT	08	USA
REC_TYPE	F	USA
MULTI_UNIT	Y	USA
GEOGRAPHIC_RANK	1	USA
DPV_CONF	Y	USA
DPV_CMRA	Y	USA
DPV_FP	Y	USA
DPV_FN1	AA	USA
DPV_FN2	AA	USA
DPV_FN3	AA	USA
DEFAULT_FLAG	Y	USA
LACSLINK_INDICATOR	Y	USA
LACSLINK_RESULT_CODE	A	USA



# Deploying Applications

This chapter discusses how to deploy your application as a Web servlet and how you can integrate MapMarker Plus 14 USA with other MapMarker country geocoders.

## In this chapter:

- ◆ **Deploying MapMarker as a Servlet** .....172
- ◆ **Integration with MapMarker for Other Countries** .....172

# 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 Apache Tomcat Servlet Container

MapMarker is installed in the Apache Tomcat 5.5 servlet container. The installer provides this environment so that you can run the servlet and sample application immediately after finishing the installation. The servlet is installed as part of the Web Application feature. Simply use the default shortcut to start the server.

The default shortcut is located where you chose to place the shortcut during installation. On Windows this is typically the Windows Start menu. The shortcut is called Start MapMarker USA v14 Server.

## Deploying MapMarker Server in Other Web Environments

MapMarker is verified and supported on Apache Tomcat, Oracle 10G App Server, BEA WebLogic, and WebSphere. See your MapMarker Release Notes for information on supported versions.

To help you with deployment into other web environments, we provide a web application archive (WAR) of the installed Web Application. This file (mapmarker40.war) is installed with the Web Application feature and is located in the sdk\tomcat\webapps directory of the Install directory.

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 read the documentation on the Sun Web site:

<http://java.sun.com/products/servlet/>

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://tomcat.apache.org/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 (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 `sdk/tomcat/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, 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 Web Application feature 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 14.0 is built upon the MapMarker Java core version 4.0. Older core versions (2.0 and 3.0) cannot be integrated into one servlet context with MapMarker 14.0.



# Custom User Dictionaries

MapMarker supports the creation and use of custom User Dictionaries based on your own source data. A User Dictionary can be used independently or as a supplement to the supplied Address Dictionary.

This chapter includes information on creating User Dictionaries, source data requirements and required fields, and other information specific to working with User Dictionaries.

As part of the Desktop Application, MapMarker provides a User Dictionary wizard to simplify user dictionary creation in the desktop application.

MapMarker provides the MapInfo User Dictionary Utility to guide you through the User Dictionary creation process. This graphical creation tool simplifies the steps for creating, configuring, and deploying User Dictionaries on the MapMarker server.

## In this chapter

- ◆ **What Is a Custom User Dictionary? . . . . .176**
- ◆ **User Dictionary Capabilities and Requirements . . . . .176**
- ◆ **User Dictionary File Names and Formats . . . . .178**
- ◆ **Additional User Dictionary Considerations . . . . .179**
- ◆ **Creating a Custom User Dictionary . . . . .181**
- ◆ **Using the MapInfo User Dictionary Utility . . . . .185**
- ◆ **Configuring the User Dictionary . . . . .187**
- ◆ **Preferring User Dictionary Matches . . . . .191**

# What Is a Custom User Dictionary?

A custom User Dictionary is a table of streets and address ranges that you use as a source for geocoding. If you have newer or more precise data than what is available in the MapMarker Address Dictionary, creating a dictionary with this data can help you get more accurate geocoding results. For example, if you have address point data you can create a User Dictionary that enables you to take advantage of the MapMarker address point interpolation capabilities.

A User Dictionary can be used by itself to geocode records, or can be used in combination with the supplied Address Dictionary.

## User Dictionary Capabilities and Requirements

The capabilities of User Dictionaries and the basic requirements for creating them are as follows.

- All fields supported by normal street geocoding can be included in User Dictionaries.
- Points Of Interest (POI) geocoding is supported in User Dictionaries. Postal or geographic centroid geocoding are not supported in User Dictionaries.
- User Dictionaries support address browsing using partial street names or Points of Interest.
- An Address Dictionary is required to create the User Dictionary. This is because the MapMarker Address Dictionary has some internal structure that must be available when you are creating a User Dictionary.

The results from a User Dictionary are similar to that from the MapMarker Address Dictionary. The only difference appears in the georesult code. A candidate from a User Dictionary has a 'U' at the end of the georesult code. A candidate that comes from the MapMarker Address Dictionary has an 'A' at the end of the georesult code.

For example: S5HPNTSCZA is a result code that comes from an Address Dictionary, while S5HPNTSCZU comes from a User Dictionary.

See the on Result Codes chapter in the *MapMarker Plus Version 14 User Guide* for a complete description of result codes.

## Source Data Requirements

The source data for User dictionaries includes street data but can also include place names and intersections.

To create a User Dictionary, your source data must conform to the following requirements:

- Source records must include required fields, and these fields are mapped during the User Dictionary creation process. If a value of a required field is empty for a particular record, then that record will not be imported into the User Dictionary. Required fields may vary for different countries. The MapInfo table must contain specific fields, which MapMarker then uses to convert the table into the dictionary format. These input fields are described in [Required Input Fields on page 177](#).
- Source records must be in a MapInfo table (.TAB file). The TAB file requirements will vary for different countries.

- Segments that make up intersections must have one or more endpoints in the intersection for MapMarker to recognize it as an intersection.
- Source records can be either point objects or segments.
- Each row in the table is equivalent to a street segment.

## Required Input Fields

You must specify the field names in the MapInfo table (.TAB file) 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, but are strongly recommended because there may be negative consequences if they are omitted. This is described in [Optional \(Recommended\) Input Fields on page 177](#). If any of the required fields are missing, a missing field error code is returned.

The following table describes the required input fields.

Required Fields	Description	Maximum Field Length
Left Start Address	Start of address range on left side of street	10
Right Start Address	Start of address range on right side of street	10
Left End Address	End of address range on left side of street	10
Right End Address	End of address range on right side of street	10
Street Name	Name of street	30
State Abbreviation	Two-character state abbreviation	2
Left ZIP Code	ZIP Code™ for left side of street	5
Right ZIP Code	ZIP Code for right side of the street	5

## Optional (Recommended) Input Fields

The Left and Right Odd/Even Indicator fields are used to specify whether the sides of the street segment contain odd or even address ranges. Although these indicators are not required for creating a User Dictionary, it is important to use the Odd/Even Indicators when your data contains odd/even address numbers.

When the Odd/Even Indicator is specified, but is inconsistent with address numbers, the indicator is set to Both.

When the Odd/Even Indicator is **not** specified and both Start Address and End Address have values, the indicator is set to Both, unless the start and end address numbers are the same number. In that case, the indicator is set to Odd if the address numbers are odd, and set to Even if the address numbers are even.

## User Dictionary File Names and Formats

When the Odd/Even Indicator is **not** specified and both Start Address and End Address have values, the indicator is set to Both (odd and even).

**Note** If your table contains odd/even indicator information, we strongly recommend that you use the Odd/Even indicator fields. These fields ensure that your geocoded addresses are located on the correct side of the street. Omitting the fields when your data contains odd/even information may produce incorrect results.

The following table describes the optional input fields.

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

\* These fields are highly recommended.

## User Dictionary File Names and Formats

MapMarker has some requirements for User Dictionary files that you must be aware of before you create a User Dictionary:

- Each User Dictionary has a base name of eight characters or fewer.
- Each User Dictionary resides in its own directory.
- The maximum length of a path to a User Dictionary is 1024 characters.
- The maximum length of the paths to multiple User dictionaries is 1024 characters.
- The ZIP Code™ range in the MapInfo table for a User Dictionary is unlimited.

Because each User Dictionary resides in its own directory, User dictionaries may share the same name. However, it is generally good practice to use a unique name for each User Dictionary.

Some of the output files are tied to the base name. The other output files have constant names. For example, the output files for a dictionary called ud1 are the following:

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

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

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

The dictionary also contains these log files:

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

## Additional User Dictionary Considerations

Consider the following when working with custom User dictionaries.

**Data Access License, p. 179**

**CASS Standards, p. 179**

**Address Range Order, p. 180**

**Street Intersections and Customized User Dictionaries, p. 180**

### Data Access License

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

### CASS Standards

You cannot geocode to CASS standards using a custom User Dictionary. This also means that the ParcelPrecision Dictionary cannot be used during CASS geocoding.

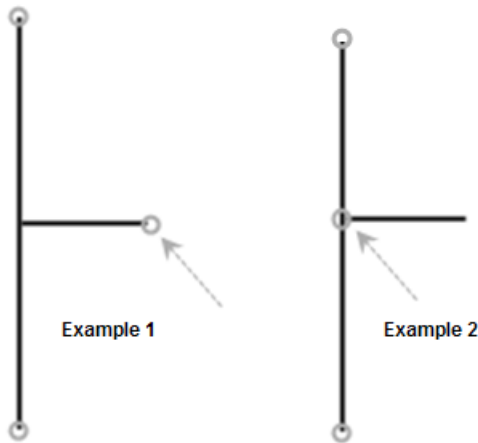
## Address Range Order

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

- If the end is greater than the start, the range is ascending
- If the start is greater than the end, the range is descending
- If the start is equal to the end, the range is ascending

## Street Intersections and Customized User Dictionaries

When geocoding to street intersections with a custom User Dictionary, MapMarker cannot recognize the intersections if one or more of the segments that make up the intersection does not have an endpoint at the intersection. This can happen when you create the User Dictionary from a customized street table in which some segments that terminate at intersections do not have endpoints (Example 1).



Example 1: Intersection in custom User Dictionary does not have endpoints for all segments. MapMarker does not recognize this as an intersection.

Example 2: Intersection in TIGER-based Address Dictionary includes endpoints for all segments. MapMarker geocodes to this intersection.

## Creating a Custom User Dictionary

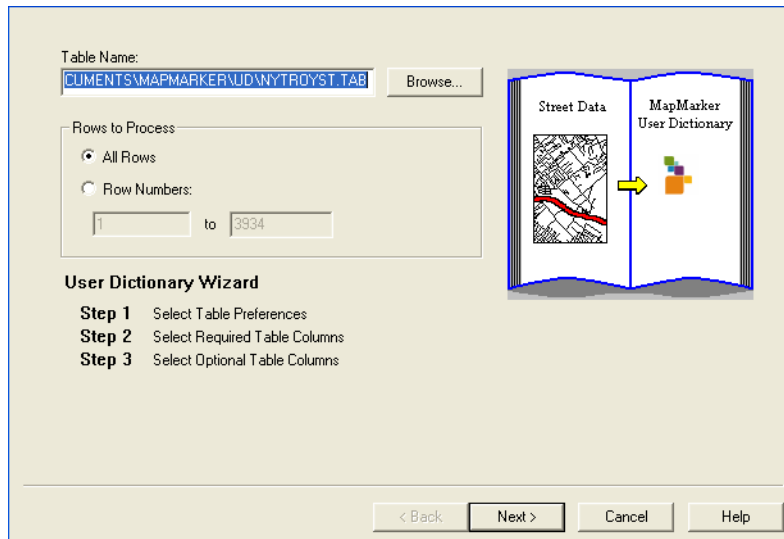
In the MapMarker Desktop Application, use the User Dictionary Wizard to create a User Dictionary.

### Using the MapMarker User Dictionary Wizard

The Desktop Application includes a wizard to walk you through the user dictionary creation process.

To create a user dictionary:

1. Choose **File > Create User Dictionary**. The User Dictionary Wizard (Step 1 of 3) dialog appears.



2. Specify the path and filename of the MapInfo table from which address information is to be derived. Alternatively, click the **Browse** button to reach the desired location of the table.
3. Specify how much of the table you wish to include in the dictionary by specifying the row numbers or leave All Rows selected.
4. Click **Next** to continue. The User Dictionary Wizard (Step 2 of 3) dialog appears.

Table: nyrtoyst.tab

OBJECTID	Required Columns	
L_F_ADD	Left Starting Address Number	Right Starting Address Number
L_T_ADD	<input type="text" value="L_F_ADD"/>	<input type="text" value="R_F_ADD"/>
R_F_ADD	Left Ending Address Number	Right Ending Address Number
R_T_ADD	<input type="text" value="L_T_ADD"/>	<input type="text" value="R_T_ADD"/>
FULLNAME	Street Name	State
POSTAL_L	<input type="text" value="FULLNAME"/>	<input type="text" value="STATE"/>
POSTAL_R	Left ZIP Code	Right ZIP Code
STATE	<input type="text" value="POSTAL_L"/>	<input type="text" value="POSTAL_R"/>
FIPS_L		
FIPS_R		
MM_STREET		
MM_GEORESULT		
MM_STD_PLACE		
LeftOE		
RightOE		
<CLEAR>		

Click and drag column names from the list into the boxes. Use <CLEAR> to clear the contents of a box. To remove a selection, select the column name <CLEAR> and drag it to the desired box.

< Back   Next >   Cancel   Help

- Specify the required fields in the street table by highlighting a field name in the list box and dragging it to the appropriate box in the address groups. Note that you must fill in all the fields in the dialog.  
When selecting the state field choose the field that contains the two-letter state abbreviation. Do not use the 2-digit FIPS code field as the state field.
- To clear any box, highlight <CLEAR> in the list and drag it over the column name you wish to delete.
- Click **Next** to continue. The User Dictionary Wizard (Step 3 of 3) dialog appears.

Table: nyrtoyst.tab

OBJECTID	Optional Columns	
L_F_ADD	Left ZIP+4	Right ZIP+4
L_T_ADD	<input type="text"/>	<input type="text"/>
R_F_ADD	Left Census Block	Right Census Block
R_T_ADD	<input type="text" value="FIPS_L"/>	<input type="text" value="FIPS_R"/>
FULLNAME	Left Odd/Even Indicator	Right Odd/Even Indicator
POSTAL_L	<input type="text" value="LeftOE"/>	<input type="text" value="RightOE"/>
POSTAL_R	Place Name	City Name
STATE	<input type="text" value="MM_STD_PLACE"/>	<input type="text" value="MM_STD_PLACE"/>
FIPS_L		
FIPS_R		
MM_STREET		
MM_GEORESULT		
MM_STD_PLACE		
LeftOE		
RightOE		
<CLEAR>		

Click and drag column names from the list into the boxes. Use <CLEAR> to clear the contents of a box. To remove a selection, select the column name <CLEAR> and drag it to the desired box.

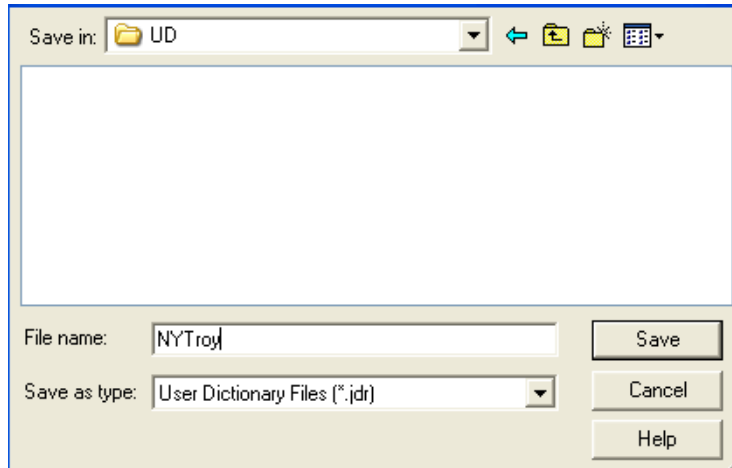
< Back   Finish   Cancel   Help

- Specify any optional fields for City, ZIP + 4<sup>®</sup>, Census Block, Odd/Even Indicator, or place name columns you wish to include in the dictionary. Including the Odd/Even Indicator allows for better placement of address points when geocoding. A place name is a point object, rather than a street segment. Examples include Sears Tower, Wrigley Field, or City Hall.

9. Click **Finish** to proceed or **Back** to revisit the previous dialogs. When you click **Finish** the Save User Dictionary As dialog appears.
10. Specify the name and location of the User Dictionary. and save in its own folder. For example, you could create a UD folder under

C:\Program Files\MapInfo\MapMarker\_USA\_v14\data\TroyUD

Save the newly created User Dictionary in this folder. This folder must be initially be empty.



Click **Save**. When the progress bar indicates that the User Dictionary is created, click **Done**.

## Adding the User Dictionary to the Desktop Application

After creating a User Dictionary, you must add the dictionary so that it can be used by the Desktop Application.

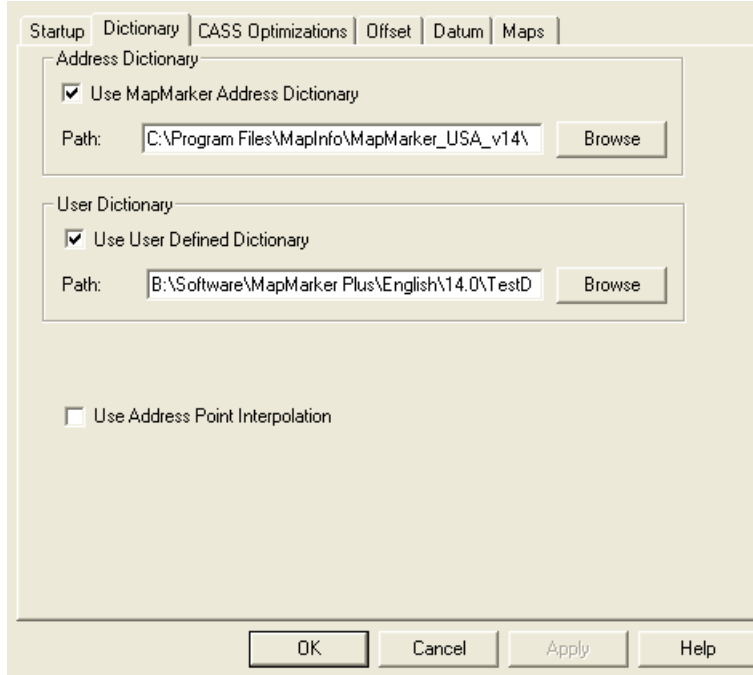
You can also specify system preferences for:

- User Dictionary (or multiple User Dictionaries) alone
- Standard MapMarker Address Dictionary
- A combination of Address Dictionary and User Dictionaries

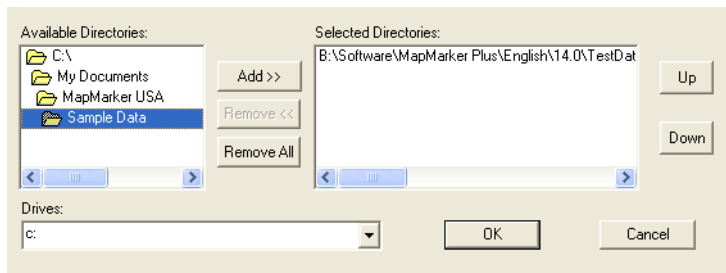
In the Desktop Application, these preferences are found in the Dictionary tab of the System Preferences dialog. If you select the User Dictionary, you can also control the search priority of User Dictionaries (if dictionaries overlap in their coverage.)

## Creating a Custom User Dictionary

1. Choose **Options > System Preferences** and select the **Dictionary** tab. Select the User Dictionary check box and **Browse** for the directory in which you created the User Dictionary. **Add** this directory to the Selected Directories column and click **OK**,



2. Check **Use User Defined Dictionary** to enable the User Dictionary.
3. Optionally, click the **Browse** button associated with the User Dictionary and adjust the search priority order of User Dictionaries. In the Choose Directories dialog, use the **Up** and **Down** arrow keys to change priorities. User dictionaries at the top of the list are given higher priority. This is important if two or more User Dictionaries overlap in coverage. For example, you may have two User Dictionaries that provide coverage of the same city, but one dictionary provides point coverage and the other dictionary offers address coverage only. In the following example, there is only one User Dictionary, so no change is required.



- On the Dictionary tab of the System Preferences dialog, you can optionally check **Use Address Point Interpolation**. This is useful if you are using a point User Dictionary (such as a ParcelPrecision Dictionary) and you want to take advantage of address point interpolation.

**Note** You can also create a User Dictionary programmatically using 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.

## Using the MapInfo User Dictionary Utility

The User Dictionary Utility can be installed as one of the available SDK features with your MapMarker Server product. This is an easy-to-use utility that simplifies the steps for creating User Dictionaries.

Once you have installed the MapInfo User Dictionary Utility and prepared your source data to meet the requirements for a User Dictionary (see [User Dictionary Capabilities and Requirements on page 176](#)), you can use the Utility to automate the User Dictionary creation process.

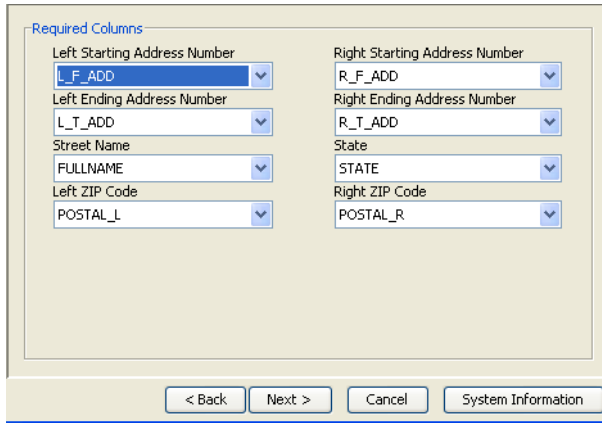
**Note** Remember that in order to create a User Dictionary, a MapMarker Address Dictionary must already be present. If you have installed any current version of MapMarker, the associated Address Dictionary is also installed.

- Start the MapMarker Plus User Dictionary Utility. If you installed this utility, it will be listed in the MapMarker USA v14 program group.
- In the Mapinfo User Dictionary Utility dialog, select USA from the Country drop-down list. In the Address Dictionary File text box, accept or Browse to the Address Dictionary file for the selected country. In the Table Name text, browse to the TAB file that contains the source data for your User Dictionary.

Click **Next** to continue.

- Select the required columns.

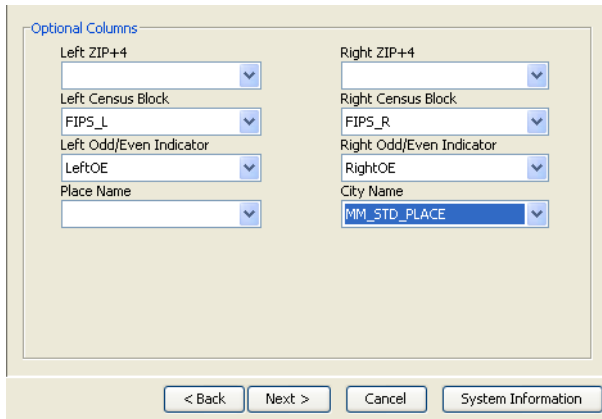
**Note** The specific field labels and details will vary for different countries. See your country-specific MapMarker Developer Guide for information on the TAB file requirements for custom user data. This will be in the chapter named Creating and Using Custom Dictionaries.



The 'Required Columns' dialog box contains two columns of dropdown menus. The left column includes: Left Starting Address Number (L\_F\_ADD), Left Ending Address Number (L\_T\_ADD), Street Name (FULLNAME), and Left ZIP Code (POSTAL\_L). The right column includes: Right Starting Address Number (R\_F\_ADD), Right Ending Address Number (R\_T\_ADD), State (STATE), and Right ZIP Code (POSTAL\_R). At the bottom are buttons for '< Back', 'Next >', 'Cancel', and 'System Information'.

Click **Next** to continue.

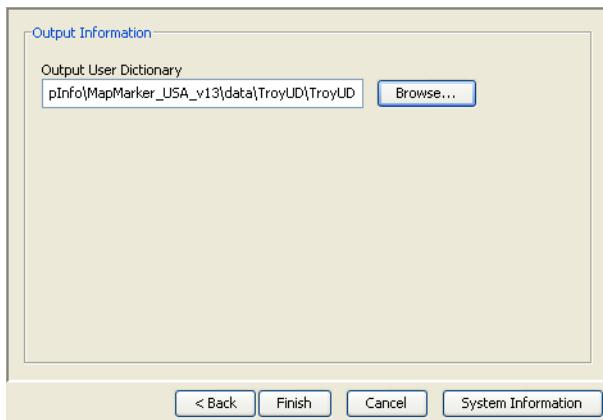
4. Select the optional columns. Since these are optional fields, you do not have to specify them.



The 'Optional Columns' dialog box contains two columns of dropdown menus. The left column includes: Left ZIP+4, Left Census Block (FIPS\_L), Left Odd/Even Indicator (LeftOE), and Place Name. The right column includes: Right ZIP+4, Right Census Block (FIPS\_R), Right Odd/Even Indicator (RightOE), and City Name (MM\_STD\_PLACE). At the bottom are buttons for '< Back', 'Next >', 'Cancel', and 'System Information'.

Click **Next** to continue.

5. Specify the output directory and file name for the User Dictionary that you are creating. You must specify an empty directory. For example, you could browse to MapMarker\_USA\_v14\data and create an empty directory named TroyUD, and in that folder save the file TroyUD.jdr.



The 'Output Information' dialog box features a text input field containing the path 'pInfo\MapMarker\_USA\_v13\data\TroyUD\TroyUD' and a 'Browse...' button. At the bottom are buttons for '< Back', 'Finish', 'Cancel', and 'System Information'.

Click **Finish** to create the User Dictionary.

6. The UD Creation Complete dialog displays a log file that summarizes processing and lists all the created files. If any errors occurred, these will be indicated in the Error File area of the dialog.
7. Click **Exit** to exit the MapInfo User Dictionary Utility.  
After successfully creating the User Dictionary, the specified directory will contain the files that comprise the User Dictionary (with .mmj, .jdr, .jdx, .sdx, and .bdx file extensions). The log file and error file are also stored in the same directory.
8. Edit the USA\_DataManagerSettings.properties file and restart the MapMarker server before you can use the User Dictionary. See [Configuring the User Dictionary on page 187](#).

At any time, you can click **System Information** for a summary of system-related information on the MapInfo User Dictionary Utility. This identifies the JVM, MapMarker core engine, and other system details.

**Note** You can also create a User Dictionary programmatically using 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.

## Configuring the User Dictionary

You can configure the USA\_DataManagerSettings.properties file so that MapMarker can find the User Dictionary and determine its ranking. There can be several instances of the properties file, depending on your installation choices.

Default locations and use of the USA\_DataManagerSettings.properties file are:

Properties File Location	Use and Configuration
<code>&lt;install&gt;\desktop\</code>	This instance of the USA_DataManagerSettings.properties file is present if you installed the desktop application. If you have written your own application using the C Developer API, you must configure this file so that MapMarker can determine the location and ranking of the User Dictionary.
<code>&lt;install&gt;\sdk\engine\lib\client\</code>	This instance of the USA_DataManagerSettings.properties file is present if you installed the Developer Kit (SDK). If you developed your own Java Application, (or are using the provided MapMarker Sample Application), you should configure this file so that MapMarker can determine the location and ranking of the User Dictionary.

Properties File Location	Use and Configuration
<code>&lt;install&gt;\sdk\ tomcat\webapps\mapmarker40\ WEB-INF\classes\ </code>	This instance of the <code>USA_DataManagerSettings.properties</code> file is used by the tomcat web server. This is used for running the provided Java Sample Application. Also, if you have written your own application using the Java API, you must configure this file so that MapMarker can determine the location and ranking of the User Dictionary.
<code>&lt;install&gt;utilities\ </code>	Exists if you installed the DPV <sup>®</sup> or LACS <sup>Link</sup> <sup>®</sup> data set. This is not relevant for User Dictionary configuration.

If you created the User Dictionary with the User Dictionary Utility (provided with the Developer Kit), you must configure the appropriate `USA_DataManagerSettings.properties` files to specify the location User Dictionary and the dictionary ranking (that is, which dictionary is used first when geocoding.)

**Note** When using the MapMarker Desktop Application CASS™ geocoding, the User Dictionary preference is disabled, except when the Address Point Interpolation preference is enabled. In that case, Address Point Interpolation is applied to the Address Dictionary only (not a User Dictionary).

When using the API directly with a User Dictionary, CASS geocoding will block the return of any candidates that do not have ZIP + 4<sup>®</sup> centroids.

See the following topics:

- [Configuring the `DataManagerSettings.properties` File, p. 188](#)
- [Setting Dictionary Preferences Using MapMarker Java API, p. 189](#)

## Configuring the `DataManagerSettings.properties` File

By default the `USA_DataManagerSettings.properties` file appears as follows:

```
# Optional - The number of dictionaries to be loaded.  DEFAULT=1
DICTIONARY_COUNT=1

# Required - The path to the highest ranking dictionary.
# Note that DICTIONARY_PATH is required from 1 to DICTIONARY_COUNT.
DICTIONARY_PATH1=/C:/Program
Files/MapInfo/MapMarker_USA_v14/MapMarker_USA_v14_Data
MAP_MARKER_HOME=/C:/Program
Files/MapInfo/MapMarker_USA_v14/MapMarker_USA_v14_Data
MAP_MARKER_SERIAL_NUMBER=GSUWEU1400123456
setting.street.data.cache.count=133
setting.street.index.cache.count=133
```

The following example shows a `USA_Datamanager.properties` file that has been edited to allow for two dictionaries (`DICTIONARY_COUNT=2`) and to add a path to the User Dictionary (`DICTIONARY_PATH2=`). The User Dictionary has a higher ranking, since it is designated as `PATH1`. In this example, the standard Address Dictionary has been configured as `PATH2`.

```
# 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=/C:/Program Files/MapInfo/MapMarker_USA_v14/
MapMarker_USA_v14_Data/Custom1

DICTIONARY_PATH2=/C:/Program Files/MapInfo/MapMarker_USA_v14/
MapMarker_USA_v14_Data
MAP_MARKER_HOME=C:\Program Files\MapInfo\MapMarker_USA_v14\
MapMarker_USA_v14_Data

```

To indicate which dictionaries to use, edit the configured dictionaries in the `USA_DataManagerSettings.properties` file. Add `PATH` keys for the dictionaries that 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 ranking of the configured dictionaries, change the rank (`DICTIONARY_PATH#`) as necessary. The integer in `PATH#` represents the order in which the dictionary is searched, where `#` can be an integer from 1 to `N`. Dictionary number 1 (`PATH1`) has the highest ranking. The number of `DICTIONARY_PATH` entries in the properties file must match the value of `DICTIONARY_COUNT`.

Close matches from lower ranked dictionaries are demoted to non-close matches. When there are close match candidates from different dictionaries, the close matches that are returned from the higher ranking dictionary are given precedence.

**Note** Remember to shut down and restart the MapMarker server after editing the `DataManagerSettings.properties` file. You must do this to make the new User Dictionary available.

## Setting Dictionary Preferences Using MapMarker Java API

The developer can also use the MapMarker Java API to specify the order of dictionaries and which dictionaries to use in a geocode request. The API settings overrides preferences set through the Desktop Application or the `DataManagerSettings.properties` file.

Dictionary preferences are 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 Dictionary search order
- Whether this dictionary is searched 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.

## Configuring the User Dictionary

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. So 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:

Method	Description
<code>getDictionaryCount</code>	Returns the number of configured dictionaries available to the engine.
<code>getDictionaryDescription</code>	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 <code>dictionarydes.txt</code> . Then place the file in the dictionary path.
<code>getSearchOrderForDictionary</code>	Returns the dictionary search order.
<code>isDictionaryAvailableForSearch</code>	Returns true if dictionary will be searched.
<code>isUserDictionary</code>	Returns true if the dictionary is a user-created dictionary.
<code>setDictionaryAvailableForSearch</code>	Setting to false indicates that the dictionary will not be searched.
<code>setSearchOrderForDictionary</code>	Sets the order in which the dictionaries will be searched.

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

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

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

**Note** For detailed information about all classes, refer to the API documentation (Javadocs) for `MapMarker`. See [Using the JavaDocs API Documentation on page 118](#).

## Preferring User Dictionary Matches

You can geocode using:

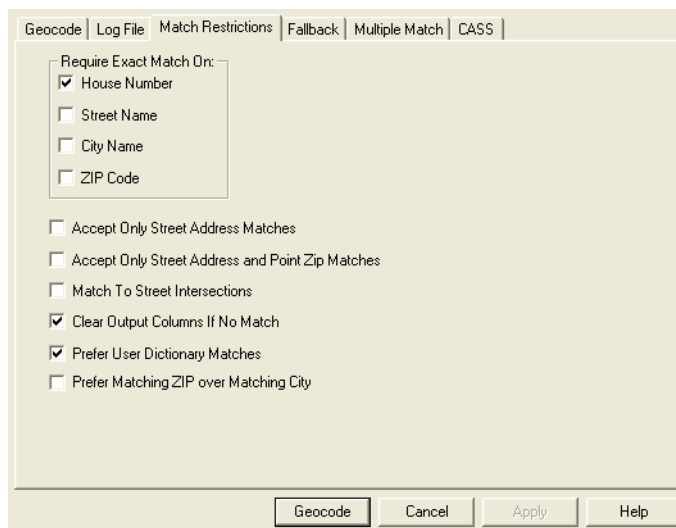
- User Dictionary (or multiple User Dictionaries) alone
- Standard MapMarker Address Dictionary
- A combination of Address Dictionary and User Dictionaries

The results from a User Dictionary are similar to that from the MapMarker Address Dictionary. The only difference appears in the georesult code. A candidate from a User Dictionary has a 'U' at the end of the georesult code. A candidate that comes from the MapMarker Address Dictionary has an 'A' at the end of the georesult code.

For example: S5HPNTSCZA is a result code that comes from an Address Dictionary, while S5HPNTSCZU comes from a User Dictionary.

See the *MapMarker Plus Version 14 User Guide* chapter on Result Codes for a complete description of result codes.

In the Desktop Application, use the Match Restrictions tab of the Geocode dialog to specify whether you prefer User Dictionary matches (or Address Dictionary matches.) If you check Prefer User Dictionary Matches, the User Dictionary will be used in preference to the Address Dictionary. Otherwise, the Address Dictionary is preferred.



If **Prefer User Dictionary Matches** is selected, the User Dictionary is used for geocoding.

If more than one User Dictionary can be used, the priority is determined in the Desktop Application by the order listed in the System Preferences. See [step 3](#) in the topic [Adding the User Dictionary to the Desktop Application](#).

With the MapMarker Server product, you can also control the priorities of deployed User Dictionaries through the `DataManagerSettings.properties` file and the MapMarker Java API.

- `USA_DataManagerSettings.properties` file: See [Configuring the DataManagerSettings.properties File on page 188](#).

## Preferring User Dictionary Matches

---

- MapMarker Java API: using the IDictionarySearchOrder object. This will override the Desktop or properties file settings. See [Setting Dictionary Preferences Using MapMarker Java API on page 189](#).

# 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** .....194
- ♦ **Country and Coordinate System Settings** .....194

# 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.

# Error Messages

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

## In this appendix:

- ♦ [MapMarker Java API Errors, p. 196](#)
- ♦ [Geocoding Engine Errors, p. 199](#)
- ♦ [Adapter Errors, p. 199](#)

## MapMarker Java API Errors

### Engine Errors

Error Code	Description
ERR_CANT_CREATE_GEOCODABLE_ADDRESS	Engine unable to create an address object for the given country.
ERR_EXCEPTION_IN_ENGINE	Engine encountered an exception.
ERR_UNKNOWN_GEOCODE_TYPE	Engine was given an unknown geocode type.
ERR_ENGINE_RESET_NULL	Engine returned null instead of a response object.

### Parser Errors

Error Code	Error #	Description
ERROR_PARSER_INPUT_ADDRESS_NULL	2000	Input address is null.
ERROR_PARSER_PARSED_ADDRESS_NULL	2001	Parsed address is null.
ERROR_PARSER_INITIALIZATION	5000	Unable to initialize parser.

### Data Access Exceptions

Error Code	Error #	Description
ERROR_DATA_ACCESS_INVALID_REQUEST	2100	Invalid request made during data access.
ERROR_DATA_ACCESS_DATA_CORRUPTED	2101	Data corruption encountered during data access.
ERROR_DATA_ACCESS_IO	2102	IO error encountered during data access.
ERROR_DATA_ACCESS_NO_DICTIONARY	2103	The data manager failed to find any usable dictionaries, based on the system preferences and settings.

Error Code	Error #	Description
ERROR_DATA_ACCESS_NO_SAC	2104	No search area codes found for given location information.
ERROR_DATA_ACCESS_NO_STREET_BASE	2105	No street base available for given street name.
ERROR_DATA_ACCESS_NO_SAC_DATA	2106	Data construction is not optimized. No data files found for given search area code: {0}.
ERROR_DATA_ACCESS_MISSING_FILE	2107	Data file missing.
ERROR_DATA_ACCESS_DATA_NOT_LICENSED	2108	Data not licensed.
ERROR_DATA_ACCESS_MISSING_FILE	5100	Missing file: {0}.

### License Exceptions

Error Code	Error #	Description
ERROR_LICENSE_INIT	2200	Unable to initialize license manager.
ERROR_LICENSE_MISSING_BASIC_LICENSE	5200	Missing required initialization license.
ERROR_LICENSE_ILLEGAL_GEOCODE_MODE	5201	Illegal geocode mode encountered during license evaluation.

### General Geocoding Process Exceptions

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

<b>Error Code</b>	<b>Error #</b>	<b>Description</b>
ERROR_GAC_DATA_MANAGER_INIT_FAILED	2306	Error initializing data manager.
ERROR_NO_IMPLEMENTATION	2307	Engine method not implemented.
ERROR_UNEXPECTED_ERROR	2308	Unexpected error encountered.
ERROR_BATCH_SIZE_EXCEEDED	2309	Maximum batch size exceeded.
ERROR_UNSUPPORTED_XML_PROTOCOL	2310	Unsupported XML Protocol request.

---

### **Client Exceptions**

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

---

### **Server Exceptions**

ERROR_SERVER_JDOM	2600	The server was unable to process the request.
ERROR_SERVER_INVALID_REQUEST_TYPE	2601	An invalid request type was provided.

---

## Geocoding Engine Errors

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

## Adapter Errors

Error Code	Error #	Description
ADAPTER_NO_CONFIG	1793	Could not find or open the config file.
ADAPTER_EXCEEDED_OPTIONS	1794	Number of options exceeds number of options specified in config file.
ADAPTER_NO_ENGINE_CLASS	1797	Unable to find mmjni.jar in classpath.
ADAPTER_NO_HELPER_CLASS	1801	Unable to find mmjni.jar in classpath.
ADAPTER_GEOCODE_RESULT_NO_CLASS	1804	Unable to find mmjni.jar in classpath.

## Adapter Errors

---

Error Code	Error #	Description
ADAPTER_CANDIDATE_NO_CLASS	1806	Unable to find mmjni.jar in classpath.
ADAPTER_RANGE_NO_CLASS	1808	Unable to find mmjni.jar in classpath.
ADAPTER_JNI_CFG_PARAMS_NO_CLASS	1810	Unable to find mmjni.jar in classpath.
ADAPTER_DOUBLE_POINT_NO_CLASS	1814	Unable to find miutil.jar in classpath.
ADAPTER_MAPMARKER_EXCEPTION_NO_CLASS	1820	Unable to find mmj.jar in classpath.
ADAPTER_MAPMARKER_FATAL_EXCEPTION_NO_CLASS	1822	Unable to find mmj.jar in classpath.
ADAPTER_NO_CONSTRAINTS_NO_CLASS	1824	Unable to find mmj.jar in classpath.
ADAPTER_NO_MM_EXCEPTION_CODES_CLASS	1826	Unable to find mmj.jar in classpath.
ADAPTER_RUNTIME_ERR	1841	Runtime error occurred.
ADAPTER_OUT_OF_MEMORY	1842	Memory exceeded. Check memory specified in config file.
ADAPTER_RUNTIME	1843	Runtime exception occurred.
ADAPTER_MAPMARKER_FATAL_UNKNOWN	1844	Unknown fatal exception occurred.
ADAPTER_UNKNOWN_EXCEPTION	1845	Unknown exception occurred.
ADAPTER_NO_JVM_DLL	1871	Unable to find jvm.
ADAPTER_NO_JVM_CREATE	1872	Unable to create jvm. Check jvm installation, or config file options.

# Index

## A

- adapter configuration file** 194
- address dictionary**
  - definition 36
  - installing 19
  - storing on hard drive 19
- address matching**
  - geocoding 10
- address point interpolation** 136
- addresses**
  - browsing 131
  - input 42–43
  - result code information 47
- airport geocoding** 135
- airports**
  - finding by code 136
  - finding by state 135

## B

- browsing addresses** 131

## C

- CAddressList** 103
- candidates**
  - definition 36
  - retrieving 125
- CASS mode**
  - constant 122
- CASS rules**
  - USA\_GeocodeConstraints 123
- CASS Standards** 179
- C-based developer tools** 50–55
- centroid**
  - definition 36
- ClearDialogText()** 66
- client installation** 30
- client/server geocoding** 52
- client/server toolkit**
  - client/server geocoding 52
  - MapMarker server 53
- close match**
  - definition 37
- closematch key** 120
- CMRA**
  - definition 37
- common schema** 146
- compiling C applications** 55–56
- Connect()** 67–68

## constant

- CASS mode 122
- match mode 122

## constraints 122

- geocoding 120

## coordinate systems

- JNI adapter configuration file settings 194
- setting in applications 124

## country settings 194

## creating

- user dictionary 176, 182, 184
- user dictionary wizard 181, 184
- Web client 129, 131

## creating .NET classes 147

## creating web client

- sample code 127, 129, 131

## creating web geocoding client 126

## custom geocoding control 58

## custom user dictionary

- creating 182

## customer service 16

## D

## data

- installing 19
- storing 19

## database development tools 52

## deployment as servlet 172

## dictionary ordering 189

## dictionary result information 47

## Disconnect() 68

## DoGeocode() 68

## DoSetProperties() 68

## DPV

- availability and status 137

## DpvGeocodeAddress() 69–70

## DpvGeocodeAddressLastLine() 70–72

## DpvGeocodeAddressLastLineWithSerial() 72–73

## DpvGeocodeAddressWithSerial() 73–74

## E

## Envinsa 35

## error message reference 196–200

## ESP SQL Server 52

## F

## fallback to geographic 120

## fallback to postal

GeocodeConstraints 120

## G

generic preferences 44

geocode requests

contents of 45

HTTP header 150

geocode types 41, 46

GeocodeAddress() 75, 78

GeocodeAddressEx() 78–79

GeocodeAddressLastLine() 79–80

GeocodeAddressLastLineEx() 81

GeocodeAddressLastLineWithSerial() 82

GeocodeAddressWithSerial() 83–84

GeocodeAddressWithSerialEx() 84

GeocodeCheckDbAvailability() 86

GeocodeConstraints 120, 124

fallback to geographic 120

fallback to postal 120

maximum candidates 120

must match 121

GeocodeFreeSet() 86

GeocodeGetCandidates() 87–88

GeocodeGetErrorText() 88

GeocodeGetHwyExitCandidate() 88

GeocodeGetServerVersion() 90

GeocodeGetStatesFound() 90

GeocodeGetStatesLicensed() 91

GeocodeHwyExit() 91

GeocodelsCandidateMultiUnit() 93

GeocodePostalCentroid() 93–94

GeocodePostalCentroidWithSerial() 94–95

geocoding

airports 135

client/server 52

geocode method 124

geographic centroids 132–133

highway exits 134

model 39–40

preferences 39

result codes 46–47

geocoding applications

choosing development tool 34–35

XML programming in .NET 146, 151

Geocoding Cartridge 52

geocoding constraints 122

geocoding from remote workstations 53

geocoding terms 36

geographic centroid geocoding 132–133

GetCandidateAt() 95

GetCandidateCensusBlockIDAt() 96

GetCandidateCityAt() 96

GetCandidateFirmAt() 97

GetCandidateLatitudeAt() 97

GetCandidateLongitudeAt() 98

GetCandidatePlus4At() 98

GetCandidatePrecisionAt() 99

GetCandidateResultCodeAt() 99

GetCandidateStateAt() 100

GetCandidateStreetAt() 100

GetCandidateZIPAt() 101

GetFullName() 101

GetName() 102

GetVersion() 102

## H

highway exit geocoding 134

HTTP headers 150–151

## I

Informix DataBlades 52

input addresses

building 42–43

setting 119

installation

client 30

multiple versions of MapMarker 28

new installation 21

removing MapMarker 29

shared 18

silent install 26

UNIX/Linux 21

upgrading 28

Windows procedure 21

integrating multiple country geocoders 172–173

intersections

customized user dictionary 180

## J

J Server

deprecation status 50

Java API

creating an application 118

Java Generic API 118

Java USA API 118

Java Virtual Machine (JVM)

adapter settings 194

javadocs location 118

JNI adapter

configuration file settings 194

OLE automation 58

running C applications 55–56

**K****key**

closematch [120](#)

**M**

**MapInfo ftp site** [16](#)

**MapInfo Professional integration** [12](#)

**MapMarker J Server**

deprecation status [50](#)

**MapMarker OCX** [52](#)

**MapMarker Plus**

sample application [142–144](#)

technical support [14](#), [16](#)

what's new [11](#)

**MapMarker requests**

HTTP headers [150–151](#)

**MapMarker server**

client/server geocoding [52](#)

geocoding from remote workstations [53](#)

request HTTP headers [150–151](#)

request time outs [55](#)

running on Windows [53](#)

with XML API [146](#), [151](#)

**MapMarker Streets**

using with MapMarker [14](#)

**MapXtreme** [35](#)

**match mode constant** [122](#)

**matching addresses**

with geographic data [10](#)

**maximum candidates**

GeocodeConstraints [120](#)

**MMJCommon.xsd** [146](#)

**MMJRequest.xsd** [146](#)

**MMJResponse.xsd** [146](#)

**modify**

silent [28](#)

**multiple country integration** [172–173](#)

**must match**

definition [37](#)

GeocodeConstraints [121](#)

USA\_GeocodeConstraints [122](#)

**N**

**NET class creation** [147](#)

**network installation** [18](#)

**new features and enhancements** [11](#)

**O**

**OLE automation API**

creating a custom control [58](#)

flow diagram [58](#)

list of OCX methods [64–66](#)

programming example [105–115](#)

summary of properties, events, methods [59](#)

**OLE automation methods**

ClearDialogText() [66](#)

Connect() [67–68](#)

Disconnect() [68](#)

DoGeocode() [68](#)

DoSetProperties() [68](#)

DpvGeocodeAddress() [69–70](#)

DpvGeocodeAddressLastLine() [70–72](#)

DpvGeocodeAddressLastLineWithSerial() [72–73](#)

DpvGeocodeAddressWithSerial() [73–74](#)

GeocodeAddress() [75](#), [78](#)

GeocodeAddressEx() [78–79](#)

GeocodeAddressLastLine() [79–80](#)

GeocodeAddressLastLineEx() [81](#)

GeocodeAddressLastLineWithSerial() [82](#)

GeocodeAddressWithSerial() [83–84](#)

GeocodeAddressWithSerialEx() [84](#)

GeocodeCheckDbAvailability() [86](#)

GeocodeFreeSet() [86](#)

GeocodeGetCandidates [87–88](#)

GeocodeGetErrorText() [88](#)

GeocodeGetHwyExitCandidate() [88](#)

GeocodeGetServerVersion() [90](#)

GeocodeGetStatesFound() [90](#)

GeocodeGetStatesLicensed() [91](#)

GeocodeHwyExit() [91](#)

GeocodeIsCandidateMultiUnit() [93](#)

GeocodePostalCentroid() [93–94](#)

GeocodePostalCentroidWithSerial() [94–95](#)

GetCandidateAt() [95](#)

GetCandidateCensusBlockIDAt() [96](#)

GetCandidateCityAt() [96](#)

GetCandidateFirmAt() [97](#)

GetCandidateLatitudeAt() [97](#)

GetCandidateLongitudeAt() [98](#)

GetCandidatePlus4At() [98](#)

GetCandidatePrecisionAt() [99](#)

GetCandidateResultCodeAt() [99](#)

GetCandidateStateAt() [100](#)

GetCandidateStreetAt() [100](#)

GetCandidateZIPAt() [101](#)

GetFullName() [101](#)

GetName() [102](#)

GetVersion() [102](#)

RefreshDialog() [102](#)

SelectCandidateAt() [102](#)

**OLE automation objects** [103–105](#)

**Oracle Geocoding Extender** [52](#)

**P****ParcelPrecision** 11**performance**

optimizing 21

**PMB**

definition 37

**positional accuracy of points**

result code information 46

**postal geocoding** 41

definition 38

**postDirectional**

definition 38

**postType**

definition 38

**preDirectional**

definition 38

**preferences**

generic and U.S. 44

geocoding 39

usage examples 44

user dictionary 189

**preType**

definition 38

**R****RefreshDialog()** 102**removing MapMarker from your system** 29**request schema** 146**request types** 168**response schema** 146**result codes**

definition 38

interpreting 46–47

retrieving 126

**running C applications** 55–56**S****sample application** 142–144**sample code**

creating web client 127, 129, 131

**schemas**

Mapmarker 146

**SelectCandidateAt()** 102**sending geocode requests** 45**servlet deployment** 172**setting**

geocoding constraints 122

input address 119

**silent install** 26**silent modify** 28**silent uninstall** 30**silent upgrade** 29**starting MapMarker Desktop Application** 31**street intersections**

customized user dictionary 180

**street level geocoding**

definition 38

**street name**

definition 38

**street-level geocoding** 41**Sun Application Server 9.0** 172**support**

customer service 16

technical support 14, 16

**T****technical support** 14, 16**Tomcat Servlet Container** 172**U****uninstall** 29

silently 30

**upgrade**

silently 29

**upgrading your installation** 28**USA\_GeocodeConstraints** 122

CASS rules 123

must match 122

user dictionary 122

**user dictionaries**

preferences 189

**user dictionary**

considerations and restrictions 179

creating 182

data requirements 176

definition 39

description 176

format requirements 178

intersections 180

USA\_GeocodeConstraints 122

**W****Web environments** 172**Web geocoding client** 128–129**web geocoding client**

creating 126

**WebLogic 9.0** 172**WebSphere 6.0** 172**Windows console application**

running MapMarker server as 54

**Windows service**

running MapMarker server as 53–54

## X

**XML API** [146](#), [151](#)

**XSD files**

creating .NET classes [147](#)

