MapInfo Geocoding Extender 2.0

for SQL Server™

USER GUIDE
# Table of Contents

Chapter 1: Introduction ................................................................. 9
   What Is Geocoding? ................................................................. 10
   MapInfo Geocoding Extender for SQL Server Overview ................. 10
      MapMarker ESP Users ......................................................... 10
         Upgrade Installations ................................................... 11
      Geocoding Extender Integration ....................................... 11
      Extender Implementation .................................................. 12
         User Defined Types ..................................................... 12
         Stored Procedures ..................................................... 12
         Triggers ................................................................. 14
      Geocoding Servers .......................................................... 14
      Microsoft SQL Server 2005 ............................................. 15
      SpatialWare Support ....................................................... 15
   Feature Summary ............................................................... 15
      Samples ................................................................. 15
   New in This Release ............................................................. 16
   Documentation ................................................................. 16

Chapter 2: Installation ............................................................... 19
   System Requirements ........................................................... 20
      Software Requirements ................................................... 20
         Administrator Rights .................................................. 20
      Supported Operating Systems ........................................ 20
      Microsoft SQL Server 2005 ............................................ 20
      MapMarker or Envinsa Geocoding Server ............................ 20
      Hardware Requirements .................................................. 21
   MapMarker ESP Users .......................................................... 21
   Before you Begin ............................................................... 21
   Setting Up User Rights in SQL Server 2005 ............................ 22
      Configuration Account .................................................... 22
      Runtime Account .......................................................... 22
   Installation ................................................................. 23
   Installation Errors ............................................................. 28
   File Locations ................................................................. 28
# Table of Contents

- **Reinstalling** ........................................... 29
- **Removing MapInfo Geocoding Extender for SQL Server** ............... 29

**Chapter 3: Working with MapInfo Geocoding Extender for SQL Server** .... 31

  - **Geocoding Overview** ........................................... 32
  - **Preparing to Geocode** ........................................... 32
  - **Geocoding Options** ........................................... 32
    - Geocode an Entire Table ........................................... 33
    - Geocode a Single Address ........................................... 33

**Chapter 4: User Defined Types** ........................................... 35

  - **Overview** ........................................... 36
  - **AddressRange** ........................................... 36
    - Attributes ........................................... 36
    - Methods ........................................... 37
  - **AddressRangeUnit** ........................................... 37
    - Attributes ........................................... 37
    - Methods ........................................... 37
  - **Candidate** ........................................... 37
    - Attributes ........................................... 38
    - Methods ........................................... 38
  - **CommonAddress** ........................................... 39
    - Attributes ........................................... 39
    - Methods ........................................... 41
  - **CommonTable** ........................................... 41
    - Attributes ........................................... 41
    - Methods ........................................... 43
  - **GeoAddress** ........................................... 43
    - Attributes ........................................... 44
    - Methods ........................................... 44
  - **InputTableInfo** ........................................... 45
    - Attributes ........................................... 46
    - Methods ........................................... 46
  - **MM_Preference** ........................................... 46
    - Attributes ........................................... 46
    - Methods ........................................... 50
  - **OutputTableInfo** ........................................... 50
    - Attributes ........................................... 50
    - Methods ........................................... 51

**Chapter 5: Geocoding a Table of Addresses** ........................................... 53

  - **Table Preparation** ........................................... 54
  - **Input Tables** ........................................... 54
Chapter 6: Geocoding a Single Address

Finding the Best Candidate

Finding a Result Set

Triggers

Using Triggers
# Chapter 7: Preference Settings

## Managing Geocoding Preferences

- **Creating a Preference**
  - Syntax: 91
  - Parameters: 92
  - Return Code Values and Results: 92
  - Example: 92
- **Deleting a Preference**
  - Syntax: 93
  - Parameters: 93
  - Return Code Values and Results: 93
  - Example: 94
- **Assigning a Default Preference**
  - Syntax: 94
  - Parameters: 94
  - Return Code Values and Results: 94
  - Example: 94
- **Getting the Default Preference**
  - Syntax: 95
  - Parameters: 95
  - Return Code Values and Results: 95
  - Example: 96
- **Setting a Preference**
  - Parameters: 96
  - Example: 97
- **Getting Preference Settings**
  - Syntax: 98
  - Parameters: 98
  - Return Code Values and Results: 98
  - Example: 98

## Using Country Tables

### Managing Country Table Settings

- **Adding a Country**
  - Syntax: 101
  - Parameters: 101
  - Return Code Values and Results: 102
  - Example: 102
- **Deleting a Country**
  - Syntax: 102
  - Parameters: 103
  - Return Code Values and Results: 103

© 2007 MapInfo Corporation. All rights reserved.
# Table of Contents

Example ................................................................. 103
Getting a Country ...................................................... 103
Syntax ............................................................... 103
Parameters ............................................................. 104
Return Code Values and Results .................................... 104
Example ................................................................. 104
Setting the Country ................................................... 105
Syntax ............................................................... 105
Parameters ............................................................. 106
Return Code Values and Results .................................... 106
Examples ............................................................... 106
Getting the Server Version .......................................... 107
Syntax ............................................................... 107
Parameters ............................................................. 107
Return Code Values and Results .................................... 107
Example ................................................................. 107

**Chapter 8: Result Codes Defined** .................................. 109

Result Codes ........................................................ 110
S Category: Single Close Match to Street Level .................. 111
Eight Additional Precision Characters .............................. 112
Z Category: Postal Code Match ...................................... 112
N Category: Non-matches ............................................ 112

**Chapter 9: Troubleshooting** ........................................ 113

Product Setup ....................................................... 114
MapMarker ............................................................. 114
Geocoding Server ..................................................... 114
MapInfo Geocoding Extender for SQL Server ...................... 114
Error Codes ........................................................... 115

Appendix A: Glossary ................................................ 119
Glossary of Terms ..................................................... 120

Appendix B: SpatialWare Support .................................... 123
Integration with SpatialWare for SQL Server 2005 .................. 124
Working with SpatialWare ............................................ 124

Appendix C: Geocoding Examples .................................. 125
Geocoding Demo Scripts .............................................. 126

Appendix D: Preference Comparison ............................... 127
Preference Comparison Table ........................................ 128
Introduction

The MapInfo Geocoding Extender for SQL Server is for users of Microsoft SQL Server who want to add SQL Server spatial points or longitude/latitude information to their address records. This is accomplished by a process known as geocoding and the results can be used for many purposes including display and spatial analysis.

In this chapter:

- What Is Geocoding? ...................................................... 10
- MapInfo Geocoding Extender for SQL Server Overview ........ 10
- Feature Summary ......................................................... 15
- New in This Release ..................................................... 16
- Documentation .......................................................... 16
What Is Geocoding?

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

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

MapInfo Geocoding Extender for SQL Server Overview

The MapInfo Geocoding Extender for SQL Server enables the geocoding of address records in a Microsoft SQL Server database through the interaction with a geocoding server. Geocoding calls can also be implemented on a “trigger” action. Triggers enable automatic geocoding of new addresses as they are added to the database, or existing addresses that have been changed. This eliminates the need to extract tables from the database, geocode them, and insert the tables back into the database.

MapMarker ESP Users

The MapInfo Geocoding Extender for SQL Server represents a major redesign of the MapMarker ESP product. It has been updated to support the most recent versions of MapMarker Plus US and Canada, international versions of MapMarker, Envinsa, and the latest Windows operating systems.

Its implementation is also different. MapMarker ESP used Extended Stored Procedures and database triggers to geocode. The Geocoding Extender makes extensive use of User Defined Type (UDT) objects, available with SQL Server 2005, for most of the geocoding functionality. Stored procedures act as an interface for the UDTs and enable you to geocode a single address or an entire table, and manage preference settings. It employs triggers to enable automatic geocoding of new or updated addresses in a table.

The table below summarizes the major differences between the components required for the older MapMarker ESP product and the new MapInfo Geocoding Extender for SQL Server.

<table>
<thead>
<tr>
<th>Software Components</th>
<th>MapMarker ESP</th>
<th>MapInfo Geocoding Extender for SQL Server</th>
</tr>
</thead>
</table>
Upgrade Installations

Because MapInfo Geocoding Extender for SQL Server is such a departure from the older MapMarker ESP product, there is no backward compatibility between the two products. If you are currently using MapMarker ESP, you will not be able to perform an upgrade installation to install the Geocoding Extender. You will need to perform a full install of the product.

Geocoding Extender Integration

The Geocoding Extender is provided as a package that integrates MS SQL Server with either the MapMarker Java or Envinsa geocoding server. The diagram below shows how the product works with SQL Server and the MapInfo geocoding servers.

<table>
<thead>
<tr>
<th>Software Components</th>
<th>MapMarker ESP</th>
<th>MapInfo Geocoding Extender for SQL Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geocoding Servers</td>
<td>MapMarker RPC Server</td>
<td>MapMarker Java Server, Envinsa Location Utility Service</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>Microsoft SQL Server v. 7.0 SP3, Microsoft SQL Server 2000</td>
<td>Microsoft SQL Server 2005, Express, Workgroup, Standard, or Enterprise</td>
</tr>
</tbody>
</table>
Extender Implementation

The implementation of MapInfo Geocoding Extender for SQL Server is more object oriented and designed to align with Microsoft SQL Server 2005 and the expanded capabilities of the most recent MapMarker geocoding engine.

User Defined Types

The main geocoding functionality in the Geocoding Extender has been encapsulated in a number of User Defined Type (UDT) objects, which are available in MS SQL Server 2005. In SQL Server 2005, each UDT corresponds to a struct or class in the .NET C# language.

Stored Procedures

The Geocoding Extender’s stored procedures are implemented using the User Defined Type objects. There are two types of stored procedures. The first type geocodes either a single address or an entire table of addresses. These stored procedures do the same geocoding tasks as certain of the Extended Stored Procedures used in MapMarker ESP. The table below gives a correspondence between the older Extended Stored Procedure and the new stored procedure implementation.
The second type of stored procedure manages preference settings. Definitions for geocoding preferences are stored in a table called mmge_preferences. These procedures enable you modify the preferences in the mmge_preferences table. They perform the same tasks as the extended stored procedures used to manage preferences in MapMarker ESP. The table below gives a correspondence between the MapMarker ESP extended stored procedure and the Geocoding Extender stored procedure implementation.

<table>
<thead>
<tr>
<th>MapMarker ESP Extended Stored Procedures</th>
<th>MapInfo Geocoding Extender for SQL Server Stored Procedures</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_findLocation</td>
<td>FindLocation</td>
<td>Geocodes a single address and returns one row. May also be used with SQL Server’s database trigger functionality.</td>
</tr>
<tr>
<td>sp_findLocationLastLine</td>
<td>Single_Fields</td>
<td>Geocodes a single address and returns one row. May also be used with SQL Server’s database trigger functionality.</td>
</tr>
<tr>
<td>sp_geocodeAddress</td>
<td>GeocodeAddress</td>
<td>Returns a result set as opposed to one row. The result set may consist of zero, one or more rows.</td>
</tr>
<tr>
<td>sp_geocodeAddressLastLine</td>
<td>GeocodeAddressLastLine</td>
<td>Returns a result set as opposed to one row. The result set may consist of zero, one or more rows.</td>
</tr>
<tr>
<td>sp_geocodeTable</td>
<td>GeocodeTable</td>
<td>Geocodes a table of addresses.</td>
</tr>
<tr>
<td>sp_geocodeTableLastLine</td>
<td>GeocodeTableLastLine</td>
<td>Geocodes a table of addresses.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MapMarker ESP Extended Stored Procedure</th>
<th>MapInfo Geocoding Extender for SQL Server Stored Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_create_preference</td>
<td>create_preference</td>
<td>Creates a preference and saves it in the mmge_preferences table.</td>
</tr>
<tr>
<td>sp_delete_preference</td>
<td>delete_preference</td>
<td>Deletes an existing preference.</td>
</tr>
<tr>
<td>sp_set_default_preference</td>
<td>set_default_preference</td>
<td>Assigns a default preference.</td>
</tr>
<tr>
<td>sp_get_default_preference</td>
<td>get_default_preference</td>
<td>Prints to the screen the name of the preference that is currently set as the default.</td>
</tr>
</tbody>
</table>
In addition, MapInfo Geocoding Extender for SQL Server also has stored procedures for managing the settings in the mmge_countries table. The country table is identified in the mmge_preferences table. These procedures include:

- add_country—add a country to specified country table.
- delete_country—delete a country from specified country table.
- get_country—returns server URL and server type of the specified country.
- set_country—sets the country/server URL association in the specified country table.
- get_product_version—returns the version and type of the geocode server used for the given country.

See Managing Country Table Settings on page 100 for information.

MapMarker ESP used a separate metadata table and had procedures for managing metadata. MapInfo Geocoding Extender for SQL Server stores metadata in the InputTableInfo and OutputTableInfo User Defined Type objects.

**Triggers**

A database trigger interface is built into MapInfo Geocoding Extender for SQL Server to enable automatic address updating and insertion. Use the findLocation set of procedures for this option.

**Geocoding Servers**

MapInfo Geocoding Extender for SQL Server works with either the MapMarker Java or Envinsa Location Utility Service.

The MapMarker Java server is provided with the MapMarker and MapMarker Plus developer products to extend functionality. It manages requests to and from the MapMarker geocoding engine, supports a queueing and multi-threading function for multiple geocoding requests, and provides a buffer for results. The MapMarker Java server also ensures the correct interface to the SQL Server database functionality, particularly in allowing the use of database triggers. Please refer to your MapMarker Developer documentation for more detailed information.

The Envinsa Location Utility Service is part of the Web services in the Envinsa Location Platform. Envinsa provides a country-independent API for the geocoding capabilities.

Both the MapMarker Java Server and the Envinsa Location Utility Service make calls to the MapMarker Java 4.0 core engine.
Microsoft SQL Server 2005

Microsoft SQL Server provides data storage, session management, and a container for stored procedures. The Extender works only with Microsoft SQL Server 2005. It does not work with earlier versions of SQL Server.

SQL Server clients can open a SQL Server session either through Open Database Connectivity (ODBC), using Microsoft tools such as Interactive SQL (isql), or db_library. The clients’ requests are passed to the geocoding server by MapInfo Geocoding Extender for SQL Server using the MapMarker 4.0 XML API or Envinsa Location Utility XML API. These requests are then submitted to, and performed by, the geocoding engine. The results are passed back through the geocoding server.

SpatialWare Support

MapInfo SpatialWare 4.9 for SQL Server is fully supported by MapInfo Geocoding Extender for SQL Server and vice versa. SpatialWare allows the storage, access, management, and manipulation of spatial data as a standard component in a data set.

When using the Geocoding Extender with SpatialWare, point data is written into separate x, y columns. SpatialWare geometry format can also be written to the table. In MapMarker ESP, this was handled by a preference. In the Geocoding Extender, a Spatial column must be defined in the Extender’s OutputTableInfo UDT object to hold the spatial objects. Users can also call the GetSQLSpatialObject method in the Candidate UDT to get the binary stream that represents the spatial point location of the geocoded candidate.

Please refer to SpatialWare Support in Appendix B on page 123 for additional information about using SpatialWare with this product.

Feature Summary

This is some of the main functionality provided with MapInfo Geocoding Extender for SQL Server.

• Specifying Parameters—Geocoding parameters may be specified and set through an interface provided with the package. These can then be defined as a set for use with a particular run or call.
• Automatic Updates—An extremely useful feature is that the Geocoding Extender allows geocoding calls to be implemented on a trigger action. This provides the automatic updating of database address information.
• Geocoding Options—You may geocode addresses individually or process an entire table using the procedures provided with the package.
• SpatialWare Integration—The Geocoding Extender also allows full integration with SpatialWare’s rich suite of spatial analysis capabilities.

Samples

To learn how to use MapInfo Geocoding Extender for SQL Server, review the samples that are installed in the MMESP\demo directory where MapMarker is installed. A summary of these samples is described in Geocoding Examples on page 125.
New in This Release

The MapInfo Geocoding Extender has been completely updated to support the latest Windows operating systems, Microsoft SQL Server 2005, and MapInfo’s most recent geocoding engine and servers. The new support and features include:

- **Updated geocoding engine**—The Geocoding Extender now communicates with the MapMarker Java 4.0 core engine. This enables you to take advantage of the capabilities in the updated engine. These include:
  - All current result codes
  - User dictionary support, including MapInfo Parcel Precision (U.S. only)
  - Geocode to firm names
  - Geocode to intersections
  - Geocode to geographic centroids
  - Delivery Point Validation available in the U.S.
- **Updated geocoding servers**—The Geocoding Extender uses the MapInfo geocoding servers that support the MapMarker Java 4.0 core engine. These are:
  - MapMarker or MapMarker Plus U.S. Developer products v. 11.0 or later,
  - MapMarker Plus Developer Canada v. 7.0 or later
  - MapMarker Spain 4.0 or later
  - Envinsa 4.0 Location Utility Service.

Envinsa supports geocoding in Canada, Germany, Spain, the United Kingdom, and the United States.
- Envinsa World City Geocoding supported.
- Microsoft SQL Server 2005 support. Note that the Geocoding Extender is not compatible with earlier versions of SQL Server.
- Internationalization—The Extender can communicate with multiple MapMarker and/or Envinsa server instances based on the country of the input address. Each country has one URL of the geocoding server associated with it.
- User Defined Type Objects—The core geocoding functionality is contained in SQL Server 2005 User Defined Type (UDT) objects. The stored procedures provide the API for the UDTs.
- Enables batch processing of records inside the database.
- Compatible with SpatialWare 4.9 or later.
- Compatible with MapInfo Geocoding Extender for Oracle (The Geocoding Extender for Oracle does not currently run with Envinsa. It is available with MapMarker.)
- Unicode support.

Documentation

Several sources of documentation are available to assist in effectively using MapInfo Geocoding Extender for SQL Server. We recommend you read the following documents carefully before using the Geocoding Extender:

- MapInfo Geocoding Extender for SQL Server User Guide: (this document) provides installation instructions, gives a product overview, and information on how to use the product.
- MapMarker Plus Developer Guide: Provides information on writing geocoding applications, the MapMarker Java API, and the MapMarker XML API.
• MapMarker Desktop User Guide: Provides information on geocoding using the MapMarker GUI, plus information on geocoding preferences, result codes, and user dictionaries.
• Envinsa Location Utility Service Reference: Provides information on geocoding, reverse geocoding, and gazeteer geocoding using Envinsa’s Location Utility Service.
• Envinsa .NET Client Developer Guide: Explains how to use the Geocoding Web Control to make geocode requests to the Envinsa Location Utility Service for .NET applications.
• Envinsa .NET Web Controls API Reference: Provides the .NET API of the Web Controls, including the Geocoder Control.
• Envinsa Java Client Developer Guide: Explains how to use the Geocoding Web Control to make geocode requests to the Envinsa Location Utility Service for Java applications.
• Envinsa Java Web Controls API Reference: Provides the Java API of the Web Controls, including the Geocoder Control.
• Envinsa Batch Geocoding Guide: Describes how to geocode a Content using the batch geocode facility available in the Content Manager.
• Microsoft SQL Server and MapInfo SpatialWare operations: See the Microsoft SQL Server and MapInfo SpatialWare documentation sets for additional information.

MapInfo product documentation is available on the MapInfo Web site at: http://www.mapinfo.com/documentation.

Note: This document assumes that you are familiar with SQL Server (and SpatialWare if used), including how to update tables, add columns and create spatial indexes (for SpatialWare).
This chapter outlines what you will need to do to install MapInfo Geocoding Extender for SQL Server, and what to do if you encounter any problems.

In this chapter:

- System Requirements ................................................. 20
- Before you Begin ......................................................... 21
- Setting Up User Rights in SQL Server 2005 .................. 22
- Installation Errors ....................................................... 28
- File Locations .......................................................... 28
- Reinstalling ............................................................. 29
- Removing MapInfo Geocoding Extender for SQL Server .... 29
System Requirements

MapInfo Geocoding Extender for SQL Server requires several components to run. The software and hardware requirements are described below:

Software Requirements

Administrator Rights

You must have Administrator rights on the installation machine to install MapInfo Geocoding Extender for SQL Server. You must also have the ‘sysadmin’ role in Microsoft SQL Server to configure SQL Server for the Geocoding Extender.

Supported Operating Systems

The Geocoder Extender for SQL Server runs on the following operating systems:

- Microsoft Windows 2000 Server with Service Pack (SP) 4 or later
- Microsoft Windows 2000 Professional Edition with SP 4 or later
- Microsoft Windows XP Professional Edition with SP 2 or later
- Microsoft Windows Server 2003 Enterprise Edition

Microsoft SQL Server 2005

You must have Microsoft SQL Server 2005 installed on the computer that you want to install the Geocoding Extender on. Earlier versions of Microsoft SQL Server are not compatible with MapInfo Geocoding Extender for SQL Server.

Your version of SQL Server must be configured with Windows and have SQL authentication enabled. The Geocoding Extender installer will create a new SQL user, MMGE.

SQL Server 2005 must have all the prerequisites to be .NET CLR enabled. The IIS server must also be installed.

MapMarker or Envinsa Geocoding Server

MapInfo Geocoding Extender for SQL Server runs with the MapMarker Java 4.0 core engine and requires a geocoding server that supports this engine to use. The server can reside on the same computer as Microsoft SQL Server, or on a different computer. The following geocoding servers are supported:

- MapMarker Java server
- Envinsa Location Utility Service

MapInfo has a number of geocoding products and components that support the MapMarker Java 4.0 core engine.

MapMarker Java server:

- MapMarker Developer US v. 11.0 or later
- MapMarker Plus Developer US v. 11.0 or later
Chapter 2: Installation

- MapMarker Plus Developer Canada v. 7.0 or later
- MapMarker Spain 4.0 or later

Envinsa Location Utility Service:
- Envinsa 4.0 or later

Envinsa supports geocoding in the following countries:
- Canada
- Germany
- Spain
- United Kingdom
- United States

MapInfo Geocoding Extender for SQL Server can work with versions of MapMarker and Envinsa that run on older core engines with some modifications. Please consult your technical support representative for assistance.

Hardware Requirements

You must have the minimum processor speed, memory, and disk space needed to run Microsoft SQL Server. Refer to your Microsoft SQL Server documentation for the specific requirements.

MapMarker ESP Users

MapMarker ESP Users must perform a full installation of MapInfo Geocoding Extender for SQL Server. There is no upgrade installation from MapMarker ESP to the Geocoding Extender.

Before you Begin

You must have Microsoft SQL Server 2005 and a geocoding server (MapMarker or Envinsa) installed to use MapInfo Geocoding Extender for SQL Server.

The following is the recommended order in which to install SQL Server, SpatialWare (if used), the geocoding server, and the Geocoding Extender.

1. Install SQL Server.
   Please refer to your SQL Server documentation for installation and setup instructions.

2. Install SpatialWare (optional). You must ensure that your database is prepared for SpatialWare. This involves creating spatial tables or spatializing existing tables, populating them, and creating R-tree indexes to enable spatial querying. Please refer to your SpatialWare for SQL Server documentation for installation and setup instructions. Please also see SpatialWare Support in Appendix B on page 123 for notes on using SpatialWare with the Geocoding Extender.
3. Install geocoding server.
   Test that the server is installed correctly and will run. Refer to the MapInfo product
documentation for installation instructions. The geocoding server does not have to be running to
install MapInfo Geocoding Extender for SQL Server.

4. Install MapInfo Geocoding Extender for SQL Server.

Setting Up User Rights in SQL Server 2005

To use the MapInfo Geocoding Extender for SQL Server, you must have the appropriate user rights.
There are two types of users:

• Configuration account: The user account to install the Geocoding Extender. This user must have
  the sysadmin server role.

• Runtime account: The user account to run the Geocoding Extender. The installer always creates
  a user, called MMGE, for the database where the MapInfo Geocoding Extender for SQL Server
  is installed. Its default password is 'pass'. To run the Geocoding Extender, you must be a
  member of the db_owner roles of the database where the Geocoding Extender is installed.

Configuration Account

The MapInfo Geocoding Extender for SQL Server installer uses the Configuration account to
connect to the SQL Server to perform the configuration. The configuration tasks include configuring
CLR parameters, creating tables, and registering CLR assemblies. This account must have the
sysadmin server role.

If you are connecting to the server using Server authentication, you can usually use the SQL Server
'sa' account. If you are connecting using Windows Authentication mode, make sure that the user
also has the sysadmin server role.

The installer may fail if the Configuration account user is not the owner of the database where the
Geocoding Extender is going to be installed. Making them the same can solve the problem.

Make sure that the database you are going to install the Geocoding Extender to has the
Compatibility level property in the SQL Server database properties set to ‘SQL Server 2005(90)’.
This is the default value. Some of the install script may fail if another setting is selected because the
lower levels cannot recognize the new syntax introduced in SQL Server 2005. The next dialog
shows the 'MMGE' database with the correct property setting:

Runtime Account

By default, MMGE should be the runtime account. You can use it to access all capabilities in the
MapInfo Geocoding Extender for SQL Server product. This MMGE user account is created and
configured by the installer. Other user accounts can also be used to run the Geocoding Extender
features as long as it is one of the members of the db_owner role in the database where the
Geocoding Extender is going to be installed. For example, the SQL Server Database User dialog
shows the correct database role membership for the 'MMGE' database:
Installation

Follow these instructions to install or reinstall MapInfo Geocoding Extender for SQL Server. Do the following:

1. Insert the MapInfo Geocoding Extender for SQL Server CD.

   A setup install wizard starts automatically, and the Welcome dialog displays on the screen. At the Welcome dialog click Next.

   ![Welcome dialog](image1)

   If the setup wizard does not start automatically, choose Run from the Windows Start menu. In the dialog type `e:\setup.exe` where `e` is the CD-ROM drive letter, or click the Browse button to find the setup.exe file.

   ![Setup dialog](image2)

   If you do not have Microsoft SQL Server installed, the following message displays on the screen:

   ![SQL Server not installed](image3)

   To perform a full installation of the product, you must have Microsoft SQL Server installed on the computer. Click Continue to install only samples and documentation. Click Quit to close the installation.

2. The License Agreement dialog displays on the screen.
After reading the License Agreement, click the I accept the terms of the License Agreement button, and then click Next.

Click Print if you would like to print the license agreement. If you select the I do not accept the terms in the license agreement button, the installation closes.

3. The Setup Type dialog displays on the screen.

Select one of the following:
- Complete—Click to install full product.
- Custom—Click to select which components you want to install.

Click Next.

If you selected Complete, go to step 5.

If you selected Custom, go to step 4.

4. The Custom Setup dialog displays on the screen.
Chapter 2: Installation

5. The Destination Folder dialog displays on the screen.

The default install location is:

C:\Program Files\MapInfo\GESS2.0\

To change the location of the MapInfo Geocoding Extender for SQL Server installation, click Change and navigate to the desired folder. Click Next when you are finished.

6. The Database Server dialog displays on the screen.
Select a database server on the local computer, either by choosing one from the drop-down list or clicking the Browse button. All the database instances that reside on your local computer are available in the drop-down list.

Next, choose your SQL Server authentication method.

**Note:** Your login must have Microsoft SQL Server Administrator rights. If your login credentials are incorrect, a login failed message will display when you click Next.

If Microsoft SQL Server does not respond, an error message will display, indicating that SQL server does not exist, or that access is denied.

Click Next.

7. The Database dialog displays on the screen.

This list shows the databases on the current instance of Microsoft SQL Sever.

Select a database from the list, and click Next.
8. The Geocoding Server dialog displays on the screen.

Select the geocoding server you want to use: MapMarker or Envinsa.

Specify the country using the ISO three-letter code for that country. If the country code is not valid, an error message will display.

Specify the URL of the server, and, if you selected Envinsa, the login and password. In Envinsa, the login and password are required. If the server URL is not valid, an error message will display.

Click Next when you are finished.

9. The Ready to Install the Program dialog displays on the screen.

Click Install.

10. An install status dialog displays on the screen, showing the progress of the installation. When the installation is complete, the InstallShield Wizard Completed dialog displays on the screen. Click Finish.
Installation Errors

Most of the errors that occur during installation are associated with SQL Server.

The most common causes of problems are:

• Microsoft SQL Server 2005 not installed–You must have SQL Server 2005 installed on the machine that you are installing the Geocoding Extender to.
• User Roles–To install the Geocoding Extender you must be a Configuration account user with the sysadmin server role. To run the Geocoding Extender, you must be a member of the db_owner roles of the database where the Geocoding extender is installed. Please see Setting Up User Rights in SQL Server 2005 on page 22.
• Microsoft SQL Server does not respond–The installer cannot connect to SQL Server because it either cannot find it or access is denied. Check that your login and password are correct.
• Country Code not valid–Check that the country code you supplied is the ISO three-letter code for that country.
• Server URL not valid–Check that the URL you supplied for the geocoding server is correct.

File Locations

The files associated with MapInfo Geocoding Extender for SQL Server can be found by default in the \GESS2.0 folder where you installed the Geocoding Extender.
Reinstalling

If you need to reinstall MapInfo Geocoding Extender for SQL Server, you can install over your current installation. Follow the instructions in the Setting Up User Rights in SQL Server 2005 section.

**Note:** When you reinstall, the mmge_preferences and mmge_countries tables will be re-created. Any modifications you have made to these tables will be lost.

Removing MapInfo Geocoding Extender for SQL Server

To remove MapInfo Geocoding Extender for SQL Server from your system:

1. Run the drop.sql script. This script removes the Geocoding Extender stored procedures. The default location is:
   
   C:\Program Files\MapInfo\GESS2.0\script\config

2. Delete the GESS2.0 folder.

3. Remove the mmge_preferences and mmge_country tables.

**Note:** The drop.sql script does not remove all the sample tables that the installer created nor the assembly .dlls. It unregisters them from SQL Server. If you are using any of the Geocoding Extender UDTs as a column type in the SQL Server tables, this script may fail. You will need to drop the affected tables and rerun the script.
Working with MapInfo
Geocoding Extender for SQL Server

In this chapter:

- Geocoding Overview ................................................................. 32
- Preparing to Geocode ............................................................... 32
- Geocoding Options ................................................................. 32
Geocoding Overview

Geocoding is a process in which a record containing address information is matched to an address with coordinates in a search dictionary. If a match is found, the geographic coordinates of the match record are assigned to the input address to complete the process.

Records are geocoded to street level. If a street level match cannot be made, MapMarker will attempt to match it to the nearest postal centroid. You can select different preferences to modify your results when geocoding. These preferences allow you to choose the settings by which you want to geocode. See Preference Settings in Chapter 7 on page 87 for more information.

MapInfo Geocoding Extender for SQL Server is made up of two types of features. The first is a set of User Defined Type (UDT) objects that provide the Geocoding Extender’s geocoding capabilities. The second is a set of stored procedures that provides an interface for the UDTs and enable you to geocode a single address, geocode a tables of addresses, and manage preferences.

Preparing to Geocode

Before you can geocode your records, there are a few things to consider. First you need to define your requirements. Do you simply want the latitude/longitude of what MapMarker thinks is the "best match"? If address elements are not the same, which of them should take precedence?

To geocode a table you may need to add columns to hold the output information you want returned. This may include, for example, candidate street, city, country subdivision (for example, state in the United States, or province in Canada), postal code, latitude, longitude, postal carrier route, and census. The source and results tables must be linked by a common (primary) key. It is recommended that the tables are related in a one-to-one relationship then each address record in the source table will then have a corresponding record in the results table connected by the key values. You must also ensure your columns are the correct type and length. Check this with the requirements of the procedure you are going to use.

MapInfo Geocoding Extender for SQL Server provides default preference settings for geocoding as described in Preference Settings in Chapter 7 on page 87. If you want to change these settings to your own options, you must do this before starting to geocode.

To increase performance, it is recommended that you sort your table by ZIP Code before geocoding. When the appropriate columns have been added and populated, you are ready to geocode.

Geocoding Options

Geocoding a database using MapInfo Geocoding Extender for SQL Server can be done in two ways: By geocoding either a table of addresses, or a single address.

Procedures are executed in the Query Analyzer in SQL Server. Please refer to your SQL Server documentation for more information on its use.
Geocode an Entire Table

To geocode an entire table (or a portion selected by a where clause), use the Table_Fields and Table_GeoAddr procedures. See Geocoding a Table of Addresses in Chapter 5 on page 53 for more information.

Geocode a Single Address

There are two sets of procedures provided to geocode a single address. To geocode a single address where only one row of information is returned (in separate fields) use the FindLocation and Single_Fields procedures. See Finding the Best Candidate on page 66 for more information. These procedures take different parameters if used for the trigger. See Triggers on page 82 for more information.

To geocode single addresses where a result set of zero, one or more rows is returned, use the Single_GeoAddr, GeocodeAddress, or and GeocodeAddressLastLine procedures. See Finding a Result Set on page 72 for more information.
The main geocoding capabilities of MapInfo Geocoding Extender for SQL Server are defined in User Defined Type (UDT) objects. The UDTs enable a more object oriented way to expose the geocoding functionality. In addition, Microsoft SQL Server 2005 provides .NET framework support in its UDTs and stored procedures. In SQL Server 2005, each UDT corresponds to a struct or class object in the .NET C# language.

In this chapter:

- Overview ..............................................................36
- AddressRange .......................................................36
- AddressRangeUnit ..................................................37
- Candidate ...............................................................37
- CommonAddress .....................................................39
- CommonTable ........................................................41
- GeoAddress ............................................................43
- InputTableInfo .........................................................45
- MM_Preference .......................................................46
- OutputTableInfo .......................................................50
Overview

The User Defined Type (UDT) objects in MapInfo Geocoding Extender for SQL Server are public objects. Objects, methods or properties can be added to your implementation, depending on your needs. Common parts can also be extracted in abstract classes or interfaces.

A user’s main interaction with the Geocoding Extender will be with the stored procedures, which take as parameters some of the UDT objects.

The naming convention of the UDT objects, public or private, follows the Camel-Case rule.

AddressRange

The Address Range object is used to hold the data for Address Candidate Ranges returned from geocoding. It implements the interface CommonAddress. It has all the attributes and methods of CommonAddress. It also has the following attributes and methods:

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowHouse</td>
<td>SqlString</td>
<td>The start house number of this range.</td>
</tr>
<tr>
<td>HighHouse</td>
<td>SqlString</td>
<td>The end house number of this range.</td>
</tr>
<tr>
<td>RangeCondition</td>
<td>SqlString</td>
<td>The odd/even indicator of this range. The possible values are:</td>
</tr>
<tr>
<td>RangeSide</td>
<td>int</td>
<td>The side of the street that the range is on. The possible values are:</td>
</tr>
<tr>
<td>Units</td>
<td>AddressRangeUnit[]</td>
<td>An array of AddressRangeUnit.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int GetUnitCount ()</td>
<td>This function returns the number of units in this range’s AddressRangeUnit array. If the set is empty, 0 is returned.</td>
</tr>
<tr>
<td>AddressRangeUnit GetUnit (int unitIndex)</td>
<td>Returns the AddressRangeUnit The start value of the input unit index is 0. An IndexOutOfRange error will be thrown if the given index is out of the boundary of the units.</td>
</tr>
</tbody>
</table>

AddressRangeUnit

The AddressRangeUnit object is used to hold the data for Address Candidate Range Units returned from geocoding. It contains specific information that may be different from the CommonAddress or AddressRange that they are associated with. It implements the interface CommonAddress. It has all the attributes and methods of CommonAddress. It has the following attributes and methods:

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowUnit</td>
<td>SqlString</td>
<td>The lower unit number of the range unit.</td>
</tr>
<tr>
<td>HighUnit</td>
<td>SqlString</td>
<td>The higher unit number of the range unit.</td>
</tr>
</tbody>
</table>

Methods

Not applicable.

Candidate

The Candidate object is used to hold the data for candidate addresses returned from geocoding. It implements the interface CommonAddress. It contains all the attributes and methods of CommonAddress. It has the following attributes and methods:
## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormattedStreet</td>
<td>SqlString</td>
<td>The formatted address line, e.g., 10 Wall St.</td>
</tr>
<tr>
<td>FormattedLocation</td>
<td>SqlString</td>
<td>The formatted location address as defined by the local country geocoder, e.g., NEW YORK, NY 10005-1971.</td>
</tr>
<tr>
<td>X</td>
<td>double</td>
<td>The x value of this location, e.g., for SRID=EPSG:4326. It is the longitude.</td>
</tr>
<tr>
<td>Y</td>
<td>double</td>
<td>The y value of this location, e.g., for SRID=EPSG:4326. It is the latitude.</td>
</tr>
<tr>
<td>SRID</td>
<td>SqlString</td>
<td>Represents the coordinate system the x and y coordinates should be interpreted in. DEFAULT: EPSG:4326, (WGS-84).</td>
</tr>
<tr>
<td>CloseMatchFlag</td>
<td>char</td>
<td>Indicates whether this is a close match candidate or not. T=close match, F=non close match.</td>
</tr>
<tr>
<td>ResultCode</td>
<td>SqlString</td>
<td>Describes the match precision of the candidate compared to the input address. For example, S5HPNTSCZA. Its format is described in Result Codes Defined in Chapter 8 on page 109.</td>
</tr>
<tr>
<td>Ranges</td>
<td>AddressRange[]</td>
<td>An array of AddressRange objects.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddressRange.GetRange (int rangeIndex)</td>
<td>Returns the AddressRange with the given index. If the rangeIndex is out of range or the set is NULL, a NULL object is returned.</td>
</tr>
</tbody>
</table>
Chapter 4: User Defined Types

CommonAddress

CommonAddress is not a UDT itself. It is the super class of the UDT classes such as GeoAddress, AddressRange, AddressRangeUnit, and Candidate. CommonAddress is the base layer for all the address interfaces in MapInfo Geocoding Extender for SQL Server. It should not be used directly.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>SqlString</td>
<td>The ISO 3166 three-letter country code.</td>
</tr>
<tr>
<td>GenericField1</td>
<td>SqlString</td>
<td>Field that holds additional address related attributes.</td>
</tr>
<tr>
<td>GenericField2</td>
<td>SqlString</td>
<td>Field that holds additional address related attributes.</td>
</tr>
<tr>
<td>GenericField3</td>
<td>SqlString</td>
<td>Field that holds additional address related attributes.</td>
</tr>
<tr>
<td>GenericField4</td>
<td>SqlString</td>
<td>Field that holds additional address related attributes.</td>
</tr>
<tr>
<td>PlaceName</td>
<td>SqlString</td>
<td>The company or firm name in the address.</td>
</tr>
<tr>
<td>PreAddress</td>
<td>SqlString</td>
<td>Information that comes before the main address, for example, 5 de Estanislao Figueras Calle, Madrid 28008 ESP. In this example, ‘de Estanislao’ is the PreAddress, ‘Figueras’ is the main street name, and ‘Calle’ is the PostThoroughfare Type.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int GetRangeCount ()</td>
<td>Returns the number of ranges in this candidate’s AddressRange set. If the set is NULL, zero is returned.</td>
</tr>
<tr>
<td>SqlBinary GetSQLSpatialObject ()</td>
<td>Returns a SpatialWare UDDT object that represents the location of this address candidate. It is defined in the SpatialWare object, ST_SPATIAL See Integration with SpatialWare for SQL Server 2005 on page 124 for details.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
</tr>
<tr>
<td>AddressNumber</td>
<td>SqlString</td>
</tr>
<tr>
<td>PreDirectional</td>
<td>SqlString</td>
</tr>
<tr>
<td>PreThoroughfareType</td>
<td>SqlString</td>
</tr>
<tr>
<td>MainAddress</td>
<td>SqlString</td>
</tr>
<tr>
<td>PostThoroughfareType</td>
<td>SqlString</td>
</tr>
<tr>
<td>PostDirectional</td>
<td>SqlString</td>
</tr>
<tr>
<td>PostAddress</td>
<td>SqlString</td>
</tr>
<tr>
<td>PostCode1</td>
<td>SqlString</td>
</tr>
<tr>
<td>PostCode2</td>
<td>SqlString</td>
</tr>
<tr>
<td>AreaName1</td>
<td>SqlString</td>
</tr>
<tr>
<td>AreaName2</td>
<td>SqlString</td>
</tr>
<tr>
<td>AreaName3</td>
<td>SqlString</td>
</tr>
<tr>
<td>AreaName4</td>
<td>SqlString</td>
</tr>
</tbody>
</table>
Chapter 4: User Defined Types

User Guide 41

© 2007 MapInfo Corporation. All rights reserved.

CommonTable

CommonTable is the super class of the UDT classes such as InputTableInfo, and OutputTableInfo. It is not a UDT itself. It is the super interface of both the InputTableInfo and OutputTableInfo UDTs. It should not be used directly.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>SqlString</td>
<td>The name of the database table.</td>
</tr>
<tr>
<td>KeyColumn</td>
<td>SqlString</td>
<td>The primary key column name.</td>
</tr>
<tr>
<td>CountryColumn</td>
<td>SqlString</td>
<td>The name of the column that holds the country code information. This column must hold the ISO3166 three-letter country code.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GenericField1Column</td>
<td>SqlDbType</td>
<td>The name of the column that holds the GenericField1 information.</td>
</tr>
<tr>
<td>GenericField2Column</td>
<td>SqlDbType</td>
<td>The name of the column that holds the Generic Field2 information.</td>
</tr>
<tr>
<td>GenericField3Column</td>
<td>SqlDbType</td>
<td>The name of the column that holds the GenericField3 information.</td>
</tr>
<tr>
<td>GenericField4Column</td>
<td>SqlDbType</td>
<td>The name of the column that holds the Generic Field4 information.</td>
</tr>
<tr>
<td>PlaceNameColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the company or firm name.</td>
</tr>
<tr>
<td>PreAddressColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the PreAddress information.</td>
</tr>
<tr>
<td>AddressNumberColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the AddressNumber information.</td>
</tr>
<tr>
<td>PreDirectionalColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the PreDirectional information.</td>
</tr>
<tr>
<td>PreThoroughfareTypeColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the PreThoroughfareType information.</td>
</tr>
<tr>
<td>MainAddressColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the MainAddress information.</td>
</tr>
<tr>
<td>PostThoroughfareTypeColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the PostThoroughfareType information.</td>
</tr>
<tr>
<td>PostDirectionalColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the PostDirectional information.</td>
</tr>
<tr>
<td>PostAddressColumn</td>
<td>SqlDbType</td>
<td>The name of the column that holds the PostAddress information.</td>
</tr>
<tr>
<td>PostalCode1Column</td>
<td>SqlDbType</td>
<td>The name of the column that holds the PostalCode1 information.</td>
</tr>
<tr>
<td>PostalCode2Column</td>
<td>SqlDbType</td>
<td>The name of the column that holds the PostalCode2 information.</td>
</tr>
<tr>
<td>AreaName1Column</td>
<td>SqlDbType</td>
<td>The name of the column that holds the AreaName1 information.</td>
</tr>
<tr>
<td>AreaName2Column</td>
<td>SqlDbType</td>
<td>The name of the column that holds the AreaName2 information.</td>
</tr>
</tbody>
</table>
The TableName and KeyColumn are required fields. If these fields are not set, no value will be read from the input table, and no action will be performed on the output table. The other fields are optional.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void SetAdditionalField (SqlString fieldKey, SqlString addField)</td>
<td>Sets the additional field value with the given key.</td>
</tr>
<tr>
<td>SqlString GetAdditionalField (SqlString fieldKey)</td>
<td>Gets the additional field column name with the given additional field key. A NULL value is returned if the key is not found.</td>
</tr>
</tbody>
</table>

### GeoAddress

The GeoAddress object is used to hold the input address elements to be geocoded. It also has fields for storing the bookkeeping information on geocoding, which includes the number of candidates, number of close candidates, etc. It implements the interface CommonAddress. It has the following attributes and methods:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AreaName3Column</td>
<td>SqlString</td>
<td>The name of the column that holds the AreaName3 information.</td>
</tr>
<tr>
<td>AreaName4Column</td>
<td>SqlString</td>
<td>The name of the column that holds the AreaName4 information.</td>
</tr>
<tr>
<td>UnitValueColumn</td>
<td>SqlString</td>
<td>The name of the column that holds the UnitValue information.</td>
</tr>
<tr>
<td>UnitTypeColumn</td>
<td>SqlString</td>
<td>The name of the column that holds the UnitType information.</td>
</tr>
<tr>
<td>AddFields</td>
<td>Hashtable</td>
<td>A set of key/value pairs. It holds the mapping between additional fields key and column name in the table.</td>
</tr>
</tbody>
</table>
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalCandidates</td>
<td>int</td>
<td>Gets and sets the total number of candidates.</td>
</tr>
<tr>
<td>CloseCandidates</td>
<td>int</td>
<td>Gets and sets the number of close match candidates.</td>
</tr>
<tr>
<td>Candidates</td>
<td>Candidates[]</td>
<td>An array of geocode candidates for this address.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>int</td>
<td>An integer code of the error. It is 0 if there is no error or warning.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>SqlString</td>
<td>The message that describes the error or warning.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void SetAdditionalField (SqlString fieldKey, SqlString addField)</td>
<td>Sets the additional field value with the given key.</td>
</tr>
<tr>
<td>SqlString GetAdditionalField (SqlString fieldKey)</td>
<td>Gets the additional field value with the given key. If the given key is not found, a NULL value is returned.</td>
</tr>
<tr>
<td>void FindLocation ()</td>
<td>This method geocodes this GeoAddress object using the default preference. If an error occurs an application error will be raised. The text of the error will contain both a geocode-engine specific error number and explanatory text.</td>
</tr>
<tr>
<td>void FindLocationWithPrefName (SqlString preferenceName)</td>
<td>This method geocodes this GeoAddress using the named preference from the MMGE_PREFERENCES table. Errors are handled in the same manner as above.</td>
</tr>
<tr>
<td>void FindLocationWithPref (MM_Preference prefObject)</td>
<td>This method geocodes this GeoAddress using the given preference. Errors are handled as above.</td>
</tr>
</tbody>
</table>
### InputTableInfo

The InputTableInfo object is used to hold the metadata information about an input table to geocode. The InputTableInfo object, with the OutputTableInfo object, replaces the metadata table in the MapMarker ESP version of the product.

InputTableInfo implements the interface CommonTable. It has the following attributes and methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoAddress GetGeocodedAddress()</td>
<td>This method copies this GeoAddress (using CommonAddress.copy(), thereby leaving out the output only fields) and then geocodes that address using the default preference. This geocoded copy will be returned. For the function versions of FindLocation, the application error will be raised, but when this happens, nothing is returned.</td>
</tr>
<tr>
<td>GeoAddress GetGeocodedAddressWithPrefName(SqlString preferenceName)</td>
<td>This method copies this address and returns the geocoded copy using the named preference. Errors are handled as above.</td>
</tr>
<tr>
<td>GeoAddress GetGeocodedAddressWithPref(MM_Preference prefObject)</td>
<td>This method copies this GeoAddress and returns the geocoded copy using the given preference. Errors are handled as above.</td>
</tr>
<tr>
<td>int IsGeocoded ()</td>
<td>This function returns 1 (TRUE) if this GeoAddress has a Candidate set, 0 (FALSE) if not.</td>
</tr>
<tr>
<td>int GetNumCandidates ()</td>
<td>This function returns the number of candidates in this GeoAddress's Candidate set. If the set is NULL, 0 is returned.</td>
</tr>
</tbody>
</table>
| Candidate GetCandidate (int candidateIndex) | This function returns the Candidate from this GeoAddress's CandidateSet at the given candidateIndex. If no argument is passed the default of 1 is used. If the index is out of range or the set is NULL, a NULL value is returned.
## MM_Preference

The MM_Preference object is used to hold the geocoding preferences and other settings for completing a geocode. This UDT corresponds to one preference record in the preferences table. It has the following attributes and methods:

### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultCountry</td>
<td>SqlString</td>
<td>If this attribute is set, all the addresses in the input table will be in this country if the country column does not have a value or the country column is not set. If this attribute is not set, the Geocoding Extender will use the default country defined in the country table referenced in the current MM_Preference.</td>
</tr>
<tr>
<td>Name</td>
<td>SqlString</td>
<td>Identifies the preference to be used in geocoding. It has the key value of the Primary Key column in the preferences table. Users can choose different preferences by either specifying its name or using its instance in the geocoding methods.</td>
</tr>
<tr>
<td>CountryTable</td>
<td>SqlString</td>
<td>The name of the country table. One MM_Preference instance points to one country table. It uses the geocoding server URL associated with the country code in the country table.</td>
</tr>
<tr>
<td>CloseOnly</td>
<td>char</td>
<td>Indicates if close match candidates only are returned from the geocoding process, or if non close match candidates are also returned. T=True, F=False</td>
</tr>
</tbody>
</table>

### Methods

Not applicable.
### User Defined Types

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxCandidates</td>
<td>int</td>
<td>The maximum number of candidates to be returned. This only applies to the geocoding process that can return more than one candidate. Set to -1 to return all the candidates that the engine found.</td>
</tr>
<tr>
<td>MaxRangesPerCandidate</td>
<td>int</td>
<td>The maximum number of address ranges per candidate. Set to -1 to return all the ranges that the engine found.</td>
</tr>
<tr>
<td>MaxRangeUnitsPerRange</td>
<td>int</td>
<td>The maximum number of range units per address range. Set to -1 to return all the units that the engine found.</td>
</tr>
<tr>
<td>ServerURL</td>
<td>SqlString</td>
<td>The geocoding server url string, e.g., <a href="http://localhost:8095/mapmarker40/servlet/mapmark">http://localhost:8095/mapmarker40/servlet/mapmark</a>. If this value is set, it will overwrite the URL specified in the country table. Otherwise, the Geocoding Extender will search the URL in the country table with the country code in the address.</td>
</tr>
<tr>
<td>BatchSize</td>
<td>int</td>
<td>The number of geocode results to be written back to the SQL Server output table. It applies to table geocoding process only. Its value can affect the performance of table geocoding.</td>
</tr>
<tr>
<td>PostalFallback</td>
<td>char</td>
<td>Indicates whether to fall back to the postal code if a street-level geocoding result cannot be found. T=fall back, F=do not fall back</td>
</tr>
<tr>
<td>GeographicFallback</td>
<td>char</td>
<td>Indicates whether to fall back to a geographic area (city level) if a street-level geocoding result cannot be found. T=fall back, F=do not fall back</td>
</tr>
<tr>
<td>CornerOffsetUnit</td>
<td>SqlString</td>
<td>The unit of the corner offset distance. Valid values are: “ft”, “mi”, “m”, “km”.</td>
</tr>
<tr>
<td>CornerOffsetDistance</td>
<td>int</td>
<td>The value of the corner offset distance.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>StreetOffsetUnit</td>
<td>strSql</td>
<td>The unit of the street offset distance. Valid values are: “ft”, “mi”, “m”, “km”.</td>
</tr>
<tr>
<td>StreetOffsetDistance</td>
<td>int</td>
<td>The value of the street offset distance.</td>
</tr>
<tr>
<td>RequestSRID</td>
<td>strSql</td>
<td>The coordinate reference system to be used in the geocoding result.</td>
</tr>
<tr>
<td>MustMatchAddressNumber</td>
<td>char</td>
<td>Indicates whether a close match candidate must have exactly the same address number as the input address.</td>
</tr>
<tr>
<td>MustMatchMainAddress</td>
<td>char</td>
<td>Indicates whether a close match candidate must have exactly the same street name as the input address.</td>
</tr>
<tr>
<td>MustMatchAreaName1</td>
<td>char</td>
<td>Indicates whether a close match candidate must have exactly the same country subdivision as the input address.</td>
</tr>
<tr>
<td>MustMatchAreaName2</td>
<td>char</td>
<td>Indicates whether a close match candidate must have exactly the same country secondary subdivision as the input address.</td>
</tr>
<tr>
<td>MustMatchAreaName3</td>
<td>char</td>
<td>Indicates whether a close match candidate must have exactly the same city name as the input address.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| MustMatchAreaName4     | char          | Indicates whether a close match candidate must have exactly the same city subdivision as the input address.  
|                        |               | T=True                                                                                         |
|                        |               | F=False                                                                                         |
| MustMatchPostal        | char          | Indicates whether a close match candidate must have exactly the same postal code as the input address.  
|                        |               | T=True                                                                                         |
|                        |               | F=False                                                                                         |
| MustMatchInput         | char          | Indicates whether a close match candidate must have all of the MustMatch fields be exactly the same as the input address.  
|                        |               | T=True                                                                                         |
|                        |               | F=False                                                                                         |
| DictionaryPreference   | SqlString     | The dictionary usage preference. The valid values are:                                           
|                        |               | • PREFER_UD                                                                                     |
|                        |               | • PREFER_AD                                                                                     |
|                        |               | • UD_ONLY                                                                                        |
|                        |               | • AD_ONLY                                                                                        |
|                        |               | • AD_AND_UD                                                                                      |
| addFields              | Hashtable     | Holds all the country-specific preferences in key/value pairs. Both the key and value are in string type, e.g., For CASS mode in the United States:  
|                        |               | Key=Key_CASS_RULES Value=True.                                                                      |
| LogFile                | SqlString     | The full path name of the log file. This applies to table geocoding only. If it is set, all the warning and error messages will be put into the log file rather than the SQL Server management studio console. |
| LogLevel               | char          | The log level. This applies to table geocoding only. The possible values are:                     
|                        |               | • E=error only;                                                                                   |
|                        |               | • W=errors and warning messages;                                                                   |

© 2007 MapInfo Corporation. All rights reserved.
OutputTableInfo

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void SetAdditionalField (SqlString fieldKey, SqlString addField)</td>
<td>Sets the additional geocoding preference field value with the given key.</td>
</tr>
<tr>
<td>SqlString GetAdditionalField (SqlString fieldKey)</td>
<td>Gets the additional geocoding preference value with the given key; a NULL object will be returned if the given key is not found.</td>
</tr>
</tbody>
</table>

OutputTableInfo

The OutputTableInfo object holds the output address information of a table. It implements the interface CommonTable. The OutputTableInfo object has the following attributes and methods:

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormattedStreet</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the FormattedStreet information.</td>
</tr>
<tr>
<td>FormattedLocation</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the FormattedLocation information.</td>
</tr>
<tr>
<td>X</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the x coordinate value of the location.</td>
</tr>
<tr>
<td>Y</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the y coordinate value of the location.</td>
</tr>
<tr>
<td>SRID</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the coordinate system information.</td>
</tr>
<tr>
<td>Spatial</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the SpatialWare UDDT object, which represents the location of this candidate address.</td>
</tr>
<tr>
<td>CloseMatch</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the information on whether the result is a close match candidate.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TotalCandidates</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the number of result candidates that were found for the address record in the input table.</td>
</tr>
<tr>
<td>TotalClose</td>
<td>SqlString</td>
<td>The name of the column in the output table that holds the number of close match result candidates that were found for the address record in the input table.</td>
</tr>
<tr>
<td>ResultCode</td>
<td>SqlString</td>
<td>The name of the column in the output table that stores the result code information.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>SqlString</td>
<td>The name of the column in the output table that stores the error code for the geocode result.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>SqlString</td>
<td>The name of the column in the output table that stores the errors or warning messages for the geocoding result.</td>
</tr>
</tbody>
</table>

**Methods**

Not applicable.
Geocoding a Table of Addresses

This chapter discusses how to geocode a table of addresses using the stored procedures supplied with MapInfo Geocoding Extender for SQL Server: Table.Fields and Table_GeoAddr.

In this chapter:

- Table Preparation ........................................54
- Table Geocoding Stored Procedures ..................55
Before you geocode a table, it must contain address fields that the geocoding server needs as input for the geocoding operation. The table that is to be geocoded is sometimes called the input table.

You must also decide where you want to put the results of the geocoding operation. The results, or output, can be in its own output table, or you can add additional fields to the input table to hold the output.

**Input Tables**

The Table_Fields and Table_GeoAddr stored procedures require that the input address data be supplied in tables.

Most of the fields in the InputTableInfo and OutputTableInfo UDTs are optional. Once they are set, they must match the columns value in the input and output tables. Otherwise, an error will be thrown.

If your input table contains fields that do not appear in the procedure descriptions, it is advisable to change your tables. The input table fields give the names of the columns that should be in your input table. You must have these columns as a minimum for the geocoding to be successful. This table information also applies to your output table, if you want to use one.

**Output Tables**

To output your results there are two choices:

- The first is to set up an output table to where your results will be returned. Your table must include a column for every piece of output information you require. You should also include a column for the primary key (called prim in this example).
- The second option is to have only one table for both input and output data. In this case, you must provide a column for every piece of output information you require in the same table you are using for your input data.

The Table_Fields procedure returns many fields from the geocoding operations. The OutputTableInfo object describes the columns in the output table where the geocode results are placed. See `OutputTableInfo on page 50` for information.

You might call the columns for your output data oFirm, oStreet, oCity, oState, oPostalcode, and oPostaladdoncode and so on to show they are output columns.

If you choose to output your results to a separate table, your output table might look like this:

<table>
<thead>
<tr>
<th>oFirm</th>
<th>oCity</th>
<th>oPostalcode</th>
<th>resultcode</th>
<th>longitude</th>
<th>latitude</th>
<th>prim</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapInfo</td>
<td>Troy</td>
<td>12180</td>
<td>S5HPNTSC-A</td>
<td>-73.6999440</td>
<td>42.68251</td>
<td>1</td>
</tr>
</tbody>
</table>

If you choose to output your results to the same table as your input table (so you only use one table not two) your single table might look like this:
For both options, the order of the columns is not important.

### Table Geocoding Stored Procedures

Both the geocoding stored procedures to geocode a table take an input_table as a required argument (this is the table you wish to geocode). You may also add other optional parameters. The output_table parameter specifies the output table, where geocoding results may be returned, if you choose to return your results to a separate table.

The where_clause parameter is included to allow you to specify only certain rows in your table to be geocoded. You may use both procedures with the default preference settings provided with MapInfo Geocoding Extender for SQL Server, but if you want to use your own settings you can do this by specifying your own preference as an input parameter.

The datatype of the primary key column in the table must be in the primary or simple types in SQL Server, for example, you cannot use the UDT type as the primary key column in table geocoding.

You must look at your data to see what form your tables take and choose the correct procedure.

### Table_Fields

Use the Table_Fields stored procedure to geocodes a table if the different elements of the address are held in separate columns. The addresses are stored in normal data types such as NVARCHAR() or char(). The column information in the input table is stored in the InputTableInfo parameter.

An example of the structure of the input table if you are using Table_Fields might be:

<table>
<thead>
<tr>
<th>Column Heading</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>NVARCHAR(256),</td>
<td>Input: The firm name.</td>
</tr>
<tr>
<td>street</td>
<td>NVARCHAR(256),</td>
<td>Input: The street portion of the address</td>
</tr>
<tr>
<td>city</td>
<td>NVARCHAR(40),</td>
<td>Input: The city portion of the address.</td>
</tr>
<tr>
<td>state (country_division)</td>
<td>NVARCHAR(40),</td>
<td>Input: The state or other country division portion of the address.</td>
</tr>
<tr>
<td>postcode</td>
<td>NVARCHAR(16),</td>
<td>Input: The postal code information of the address.</td>
</tr>
</tbody>
</table>
The following is an example input table for use with `geocodeTable`:

<table>
<thead>
<tr>
<th>firm</th>
<th>street</th>
<th>city</th>
<th>state</th>
<th>postalcode</th>
<th>postaladdoncode</th>
<th>prim</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapInfo</td>
<td>1 Global View</td>
<td>Troy</td>
<td>NY</td>
<td>12180</td>
<td>8399</td>
<td>1</td>
</tr>
</tbody>
</table>

**Syntax**

```sql
CREATE PROCEDURE Table_Fields
    @inTableInfo InputTableInfo,
    @outTableInfo OutputTableInfo,
    @preference MM_Preference = NULL OUTPUT,
    @whereClause NVARCHAR(2047) = NULL,
    @totalProcessed INT = NULL OUTPUT,
    @bufferSize INT = 10000
AS
...
```

If the preference is in NVARCHAR, which indicates the preference name, the name of the stored procedure will be `Table_FieldsWithPrefName`.

```sql
CREATE PROCEDURE Table_FieldsWithPrefName
    @inTableInfo InputTableInfo,
    @outTableInfo OutputTableInfo,
    @prefName NVARCHAR(30) = NULL OUTPUT,
    @whereClause NVARCHAR(2047) = NULL,
    @totalProcessed INT = NULL OUTPUT,
    @bufferSize INT = 10000
AS
...
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inTableInfo</td>
<td>InputTableInfo,</td>
<td>In</td>
<td></td>
<td>The Input table information.</td>
</tr>
<tr>
<td>outTableInfo</td>
<td>OutputTableInfo</td>
<td>In</td>
<td></td>
<td>The Output table information.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>In/Out</td>
<td>Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>--------</td>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>prefName</td>
<td>NVARCHAR(30)</td>
<td>In/Out</td>
<td>Optional</td>
<td>A user-defined preference name from the preference table. If it is missing or a null value, the default preference will be used.</td>
</tr>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>In/Out</td>
<td>Optional</td>
<td>An MM_Preference object. If it is missing or a null value, then the default preference as defined in preference table will be used. If the preference table does not have a default preference then the default preference from the Geocoding Engine will be used.</td>
</tr>
<tr>
<td>where_clause</td>
<td>NVARCHAR(2047)</td>
<td>In</td>
<td>Optional</td>
<td>A string to be appended to the where clause. To geocode only pre-selected rows, use the standard SQL where clause without the 'where' keyword.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(e.g., 'city = ''Albany'' and country = ''US'' or city = ''Troy'')</td>
</tr>
<tr>
<td>totalProcessed</td>
<td>Int</td>
<td>Out</td>
<td>Optional</td>
<td>The number of addresses/records that have been processed.</td>
</tr>
<tr>
<td>bufferSize</td>
<td>Int</td>
<td>In</td>
<td>Optional</td>
<td>The number of rows loaded into memory for table geocoding. Its default value is 10000. Depending on how much memory has been assigned to SQL Server, users can adjust this value. If SQL Server has less than 256 MB memory allocated (default setting), users can set a smaller value for this buffer, for example, 5000. There is no need to set a customized value if the default value works.</td>
</tr>
</tbody>
</table>
Return Code Values and Results
This procedure returns zero (0) on error or the number of addresses/records that have been successfully geocoded.

Together with the output parameter, totalProcessed, users can get the number of addresses/records that failed to be geocoded. It is the difference between these two values:

\[
totalFailed = totalProcessed - totalGeocoded
\]

Example
Sample 1: Geocode a table with the given preference and where clause.

```sql
-- Define the input table metadata.
DECLARE @inputTable InputTableInfo
SET @inputTable = CONVERT(InputTableInfo, '')
SET @inputTable.TableName = 'mmge_demo'
SET @inputTable.KeyColumn = 'prim'
SET @inputTable.PlaceNameColumn = 'firm'
SET @inputTable.MainAddressColumn = 'adr'
SET @inputTable.PostAddressColumn = 'lastline'
SET @inputTable.PostalCode1Column = 'zip'
SET @inputTable.PostalCode2Column = 'zip4'

-- Define the output table metadata.
DECLARE @outputTable OutputTableInfo
SET @outputTable = CONVERT(OutputTableInfo, '')
SET @outputTable.TableName = 'mmge_out'
SET @outputTable.KeyColumn = 'prim'
SET @outputTable.CountryColumn = ''
SET @outputTable.PlaceNameColumn = 'ofirm'
SET @outputTable.FormattedStreet = 'oadr'
SET @outputTable.AreaName1Column = 'ostate'
SET @outputTable.AreaName3Column = 'ocity'
SET @outputTable.PostalCode1Column = 'ozip'
SET @outputTable.PostalCode2Column = 'ozip4'
SET @outputTable.X = 'olong'
SET @outputTable.Y = 'olat'
SET @outputTable.ResultCode = 'result'

DECLARE @totalProcessed AS INT
DECLARE @totalSuccessed AS INT

DECLARE @mm_pref AS MM_Preference
SET @mm_pref = CONVERT(MM_Preference, '');
SET @mm_pref.MustMatchPostal = 'T'
DECLARE @whereClause AS NVARCHAR(50)
```
SET @whereClause = N'prim >= "5"'
EXEC @totalSuccessed = Table_Fields @inputTable, @outputTable, @mm_pref, @whereClause, @totalProcessed OUTPUT

SELECT @totalProcessed AS N'Total Records Processed',
      @totalSuccessed AS N'Successfully geocoded',
      @totalProcessed - @totalSuccessed AS N'Failed geocoded'
GO

Sample 2: Geocode a table with the given preference name, no where clause.

-- Define the input table metadata.
DECLARE @inputTable InputTableInfo
SET @inputTable = CONVERT(InputTableInfo, '')
SET @inputTable.TableName = 'mmge_inout'
SET @inputTable.KeyColumn = 'prim'
SET @inputTable.PlaceNameColumn = 'firm'
SET @inputTable.MainAddressColumn = 'adr'
SET @inputTable.PostAddressColumn = 'lastline'
SET @inputTable.PostalCode1Column = 'zip'
SET @inputTable.PostalCode2Column = 'zip4'

-- Define the output table metadata.
DECLARE @outputTable OutputTableInfo
SET @outputTable = CONVERT (OutputTableInfo, '')
SET @outputTable.TableName = 'mmge_inout'
SET @outputTable.KeyColumn = 'prim'
SET @outputTable.PlaceNameColumn = 'ofirm'
SET @outputTable.FormattedStreet = 'oadr'
SET @outputTable.AreaName1Column = 'ostate'
SET @outputTable.AreaName3Column = 'ocity'
SET @outputTable.PostalCode1Column = 'ozip'
SET @outputTable.PostalCode2Column = 'ozip4'
SET @outputTable.X = 'olong'
SET @outputTable.Y = 'olat'
SET @outputTable.ResultCode = 'result'

DECLARE @totalProcessed AS INT
DECLARE @totalSuccessed AS INT
EXEC @totalSuccessed = Table_FieldsWithPrefName @inputTable, @outputTable, 'INITIALPREF', NULL, @totalProcessed OUTPUT
Table Geocoding Stored Procedures

SELECT @totalProcessed AS N'Total Records Processed',
       @totalSuccessed AS N'Successfully geocoded',
       @totalProcessed - @totalSuccessed AS N'Failed geocoded'
GO

Table_GeoAddr

The Table_GeoAddr stored procedure geocodes a table when address information is held in GeoAddress objects. The addresses in the input table must be stored in the GeoAddress type.

Syntax

CREATE PROCEDURE Table_GeoAddr
    @geoAddressTable NVARCHAR(30),
    @geoAddressColumnName NVARCHAR(30),
    @keyColumn NVARCHAR(30),
    @oGeoAddressTable NVARCHAR(30),
    @oGeoAddressColumnName NVARCHAR(30),
    @oKeyColumn NVARCHAR(30),
    @preference MM_Preference = NULL OUTPUT,
    @whereClause NVARCHAR(2047) = NULL,
    @totalProcessed INT = NULL OUTPUT,
    @bufferSize INT = 10000
AS
...

If the preference is in NVARCHAR, which indicates the preference name, the name of the stored procedure will be Table_GeoAddrWithPrefName.

CREATE PROCEDURE Table_GeoAddrWithPrefName
    @geoAddressTable NVARCHAR(30),
    @geoAddressColumnName NVARCHAR(30),
    @keyColumn NVARCHAR(30),
    @oGeoAddressTable NVARCHAR(30),
    @oGeoAddressColumnName NVARCHAR(30),
    @oKeyColumn NVARCHAR(30),
    @prefName NVARCHAR(30) = NULL OUTPUT,
    @whereClause NVARCHAR(2047) = NULL,
    @totalProcessed INT = NULL OUTPUT,
    @bufferSize INT = 10000
AS
...

**Parameters**

The parameters that these procedures take are described in the following table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoAddressTable</td>
<td>NVARCHAR(30),</td>
<td>In</td>
<td></td>
<td>The name of the input table.</td>
</tr>
<tr>
<td>geoAddressColumnName</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td></td>
<td>The name of the column that has the GeoAddress object in the input table.</td>
</tr>
<tr>
<td>keyColumn</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td></td>
<td>The name of the primary key column in the input table.</td>
</tr>
<tr>
<td>oGeoAddressTable</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td></td>
<td>The name of the output table.</td>
</tr>
<tr>
<td>oGeoAddressColumnName</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td></td>
<td>The name of the column that holds the GeoAddress object in the output table.</td>
</tr>
<tr>
<td>oKeyColumn</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td></td>
<td>The name of the primary key column in the output table.</td>
</tr>
<tr>
<td>prefName</td>
<td>NVARCHAR(30)</td>
<td>In/Out</td>
<td>Optional</td>
<td>A user-defined preference name from the preference table. If it is missing or a null value, the default preference is used.</td>
</tr>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>In/Out</td>
<td>Optional</td>
<td>An MM_Preference object. If it is missing or a null value, then the default preference as defined in the preference table will be used. If the preference table does not have a default preference, then the default preference from the Geocoding Engine will be used.</td>
</tr>
</tbody>
</table>
### Table Geocoding Stored Procedures

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>where_clause</td>
<td>NVARCHAR(2047)</td>
<td>In</td>
<td>Optional</td>
<td>A string to be appended to the where clause. To geocode only pre-selected rows, use the standard SQL where clause without the 'where' keyword. (Optional) (e.g., 'city = ''Albany'' and country = ''US'' or city = ''Troy'')</td>
</tr>
<tr>
<td>totalProcessed</td>
<td>Int</td>
<td>Out</td>
<td>Optional</td>
<td>The number of addresses/records that have been processed.</td>
</tr>
<tr>
<td>bufferSize</td>
<td>Int</td>
<td>In</td>
<td>Optional</td>
<td>The number of rows loaded into memory for table geocoding. Its default value is 10000. Depending on how much memory has been assigned to SQL Server, users can adjust this value. If SQL Server has less than 256 MB memory allocated (default setting), users can set a smaller value for this buffer, for example, 5000. There is no need to set a customized value if the default value works.</td>
</tr>
</tbody>
</table>

### Return Code Values and Results

This procedure returns zero (0) on error or the number of addresses/records that have been successfully geocoded.

Together with the output parameter, totalProcessed, users can get the number addresses/records that have failed to be geocoded. It is the difference between these two values:

\[ \text{totalFailed} = \text{totalProcessed} - \text{totalGeocoded} \]
Example

Sample 1: Geocode a table with given preference and where clause.

DECLARE @totalProcessed AS INT
DECLARE @totalSuccessed AS INT

DECLARE @mm_pref AS MM_Preference
SET @mm_pref = CONVERT(MM_Preference, '');
SET @mm_pref.MustMatchPostal = 'T'

DECLARE @whereClause AS NVARCHAR(50)
SET @whereClause = N'prim >= "1"'

EXEC @totalSuccessed = Table_GeoAddr 'mmge_geoaddr', 'geoaddr', 'prim', 'mmge_geoaddr', 'ogeoaddr', 'prim', @mm_pref, @whereClause, @totalProcessed OUTPUT

SELECT @totalProcessed AS N'Total Records Processed',
     @totalSuccessed AS N'Successfully geocoded',
     @totalProcessed - @totalSuccessed AS N'Failed geocoded'
GO

Sample 2: Geocode a table with the given preference name, no where clause.

DECLARE @totalProcessed AS INT
DECLARE @totalSuccessed AS INT

EXEC @totalSuccessed = Table_GeoAddrWithPrefName 'mmge_geoaddr', 'geoaddr', 'prim', 'mmge_geoaddr', 'ogeoaddr', 'prim', 'INITIALPREF', NULL, @totalProcessed OUTPUT

SELECT @totalProcessed AS N'Total Records Processed',
     @totalSuccessed AS N'Successfully geocoded',
     @totalProcessed - @totalSuccessed AS N'Failed geocoded'
GO

More samples can be found in the demo directory after the Geocoding Extender is deployed.
Geocoding a Single Address

This chapter discusses how to geocode a single address. Different procedures are provided depending on whether you want to find the best candidate or if you want to find all the candidates in a result set.

In this chapter:

- Finding the Best Candidate ................................................. 66
- Finding a Result Set ........................................................... 72
- Triggers .............................................................................. 82
Finding the Best Candidate

MapInfo Geocoding Extender for SQL Server comes with stored procedures that enable you to geocode a single address and return only one best candidate. These procedures are:

- **FindLocation**—Geocodes a single address and returns one row. Use when the input address information resides in separate fields.
- **Single_Fields**—Geocodes a single address and returns one row. Use when the input address information is given in one line in an unparsed form.

If an address is geocoded successfully, these procedures return exactly one row. It is important to note that these procedures are designed to return only the best candidate even if multiple candidates are available. The geocoding preferences determine which candidate is the best candidate to return. Therefore, a situation may arise where a poor geocode result is returned even though it is not the best candidate.

These procedures can also be used (with different parameters) with database triggers. Triggers provide automatic updating of tables when new addresses are added or existing ones updated. Triggers also allow automatic single-address geocoding. The trigger automatically geocodes the changes when an address is updated or added to a table. See Triggers on page 82 for more information.

The FindLocation and Single_Fields stored procedures geocode a single address and return the result into variables.

**FindLocation**

Use the FindLocation procedure to geocode a single address and return one row. This procedure can also be used with the SQL Server’s database trigger functionality. It was called sp_findLocation in the MapMarker ESP version of the product.

This procedure is used when separate fields comprise the input data set. For example, firm, street, city, country subdivision, postalcode, postaladdoncode, citysubdivision and preference are all passed as separate parameters for this procedure.

**Syntax**

```sql
CREATE PROCEDURE FindLocation
    @firm NVARCHAR(256) OUTPUT,
    @street NVARCHAR(256) OUTPUT,
    @city NVARCHAR(256) OUTPUT,
    @countrysub NVARCHAR(40) OUTPUT,
    @postalcode NVARCHAR(10) = NULL OUTPUT,
    @postalcode2 NVARCHAR(10) = NULL OUTPUT,
    @country CHAR(3) = NULL OUTPUT,
    @citysub CHAR(40) = NULL OUTPUT,
    @preference MM_Preference = NULL OUTPUT,
    @longitude FLOAT = NULL OUTPUT,
    @latitude FLOAT = NULL OUTPUT,
    @resultcode NVARCHAR(20) = NULL OUTPUT
```

If the preference is in NVARCHAR, which indicates the preference name, the name of the stored procedure will be FindLocationWithPrefName.

CREATE PROCEDURE FindLocationWithPrefName
    @firm NVARCHAR(256) OUTPUT,
    @street NVARCHAR(256) OUTPUT,
    @city NVARCHAR(40) OUTPUT,
    @countrysub NVARCHAR(40) OUTPUT,
    @postalcode NVARCHAR(10) = NULL OUTPUT,
    @postalcode2 NVARCHAR(10) = NULL OUTPUT,
    @country CHAR(3) = NULL OUTPUT,
    @citysub CHAR(40) = NULL OUTPUT,
    @prefName NVARCHAR(30) = NULL OUTPUT,
    @longitude FLOAT = NULL OUTPUT,
    @latitude FLOAT = NULL OUTPUT,
    @resultcode NVARCHAR(20) = NULL OUTPUT
AS

Parameters

The parameters taken in this procedure are described below.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>NVARCHAR(256),</td>
<td>In/Out</td>
<td></td>
<td>The firm name.</td>
</tr>
<tr>
<td>street</td>
<td>NVARCHAR(256),</td>
<td>In/Out</td>
<td></td>
<td>The street portion of the address.</td>
</tr>
<tr>
<td>city</td>
<td>NVARCHAR(40),</td>
<td>In/Out</td>
<td></td>
<td>The city portion of the address.</td>
</tr>
<tr>
<td>countrysub</td>
<td>NVARCHAR(40),</td>
<td>In/Out</td>
<td></td>
<td>The state or other country division portion of the address.</td>
</tr>
<tr>
<td>postalcode</td>
<td>NVARCHAR(16),</td>
<td>In/Out</td>
<td>Optional</td>
<td>The postal code information for the address.</td>
</tr>
<tr>
<td>postalcode2</td>
<td>NVARCHAR(10),</td>
<td>In/Out</td>
<td>Optional</td>
<td>The additional postal code information, such as the ZIP +4 extension in the United States.</td>
</tr>
<tr>
<td>country</td>
<td>CHAR(3)</td>
<td>In/Out</td>
<td>Optional</td>
<td>The ISO three-letter country code of the address. (Optional) If it is missing, the address will be considered to be in the default country, which is set up in the country table.</td>
</tr>
<tr>
<td>citysub</td>
<td>char(40)</td>
<td>In/Out</td>
<td>Optional</td>
<td>The city subdivision in an address.</td>
</tr>
<tr>
<td>prefName</td>
<td>NVARCHAR(30)</td>
<td>In/Out</td>
<td>Optional</td>
<td>A user-defined preference name from the preference table. If it is missing or a null value, the default preference will be used.</td>
</tr>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>In/Out</td>
<td>Optional</td>
<td>An MM_Preference object. If it is missing or a null value, then the default preference as defined in the preference table will be used. In turn, if the preference table does not have a default preference, then the default preference from the Geocoding Engine will be used.</td>
</tr>
<tr>
<td>longitude</td>
<td>float</td>
<td>Out</td>
<td>Optional</td>
<td>The longitude value of the address location.</td>
</tr>
<tr>
<td>latitude</td>
<td>float</td>
<td>Out</td>
<td>Optional</td>
<td>The latitude value of the address location.</td>
</tr>
<tr>
<td>resultcode</td>
<td>NVARCHAR(20)</td>
<td>Out</td>
<td>Optional</td>
<td>The result code of the address.</td>
</tr>
</tbody>
</table>

**Note:** The procedure reads the supplied parameters in order. If a parameter is required, it must not
be omitted. It must either be set to NULL or left as an empty string if its type is string.

Return Code Values and Results

This procedure returns zero (0) on error or no candidate found and one (1) on success.

FindLocation returns all the address fields, x, y location, as well as the result code as the output parameters:

resultcode–This corresponds to the matchType in Envinsa and AdditionalFields (RESULT_CODE as the key) in the MapMarker server.

This stored procedure does not print out any other message unless an error occurs.

Examples

These examples are available in the demo directory.

Sample 1: Geocode a U.S. firm with default preference settings.

DECLARE @ret AS INT
EXEC @ret = FindLocation 'MapInfo', '', 'Troy', 'NY', NULL, NULL, 'USA'
SELECT @ret AS N'GeocodeResult'
GO

The results might look like this in the database:

<table>
<thead>
<tr>
<th>Frm</th>
<th>Street</th>
<th>Cty</th>
<th>Country/Subdivision</th>
<th>PostalCode</th>
<th>PostalCode2</th>
<th>Country</th>
<th>City/Subdivision</th>
<th>Longitude</th>
<th>Latitude</th>
<th>ResultCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GLOBAL VY</td>
<td>TROY</td>
<td>NY</td>
<td>12180</td>
<td>8399</td>
<td>USA</td>
<td></td>
<td>-73.723567</td>
<td>42.682894</td>
<td>55-PTS-ZA</td>
</tr>
</tbody>
</table>

Sample 2: Geocode a U.S. address with default preference.

DECLARE @prefName AS NVARCHAR(30)
DECLARE @ret AS INT
EXEC @ret = FindLocationWithPrefName NULL, '1 main st', 'New York', 'NY', NULL, NULL, 'USA', NULL, @prefName OUTPUT
SELECT @ret AS N'GeocodeResult', @prefName AS N'PreferenceName'
GO

The results might look like this in the database:

<table>
<thead>
<tr>
<th>Frm</th>
<th>Street</th>
<th>Cty</th>
<th>Country/Subdivision</th>
<th>PostalCode</th>
<th>PostalCode2</th>
<th>Country</th>
<th>City/Subdivision</th>
<th>Longitude</th>
<th>Latitude</th>
<th>ResultCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 W MAIN ST</td>
<td>NORWICH</td>
<td>NY</td>
<td>13815</td>
<td>1612</td>
<td>USA</td>
<td></td>
<td>-75.623851</td>
<td>42.531141</td>
<td>55-PTS-ZA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Geocode Result</th>
<th>Preference Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INITIALPREF</td>
</tr>
</tbody>
</table>

More samples can be found in the demo directory after the Geocoding Extender is deployed.
**Single_Fields**

The Single_Fields stored procedure geocodes a single address and returns one row. This procedure can also be used with the SQL Server's database trigger functionality. It was called sp_findLocationLastLine in the MapMarker ESP version of the product.

Use Single_Fields if the input data is given as a single line containing the address information in an unparsed form. For example, firm, street, preference and lastline (which includes city, state, countrydivision, postalcode, and postalcode2 fields) are passed as parameters for this procedure.

**Syntax**

```sql
CREATE PROCEDURE Single_Fields
    @firm NVARCHAR(256) OUTPUT,
    @street NVARCHAR(256) OUTPUT,
    @lastline NVARCHAR(256) OUTPUT,
    @country CHAR(3) = NULL OUTPUT,
    @preference MM_Preference = NULL OUTPUT,
    @longitude FLOAT = NULL OUTPUT,
    @latitude FLOAT = NULL OUTPUT,
    @resultcode NVARCHAR(20) = NULL OUTPUT
AS
...
```

If the preference is in NVARCHAR, which indicates the preference name, the name of the stored procedure will be Single_FieldsWithPrefName.

```sql
CREATE PROCEDURE Single_FieldsWithPrefName
    @firm NVARCHAR(256) OUTPUT,
    @street NVARCHAR(256) OUTPUT,
    @lastline NVARCHAR(256) OUTPUT,
    @country CHAR(3) = NULL OUTPUT,
    @prefName NVARCHAR(30) = NULL OUTPUT,
    @longitude FLOAT = NULL OUTPUT,
    @latitude FLOAT = NULL OUTPUT,
    @resultcode NVARCHAR(20) = NULL OUTPUT
AS
...
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>NVARCHAR(256)</td>
<td>In/Out</td>
<td></td>
<td>The firm name.</td>
</tr>
<tr>
<td>street</td>
<td>NVARCHAR(256)</td>
<td>In/Out</td>
<td></td>
<td>The street portion of the address.</td>
</tr>
<tr>
<td>lastline</td>
<td>NVARCHAR(256)</td>
<td>In/Out</td>
<td></td>
<td>String containing the unparsed city, state, postalcode and postaladdoncode information for the address.</td>
</tr>
<tr>
<td>country</td>
<td>CHAR(3)</td>
<td>In/Out</td>
<td>Optional</td>
<td>The ISO three-letter country code of the address. (Optional) If it is missing, the address will be considered to be in the default country, which is set up in the country table.</td>
</tr>
<tr>
<td>prefName</td>
<td>NVARCHAR(30)</td>
<td>In/Out</td>
<td>Optional</td>
<td>A user-defined preference name from the preferences table. If it is missing or a null value, the default preference will be used.</td>
</tr>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>In/Out</td>
<td>Optional</td>
<td>An MM_Preference object. If it is missing or a null value, the default preference as defined in the preference table will be used. If the preference table does not have a default preference, the default preference from the Geocoding Engine will be used.</td>
</tr>
<tr>
<td>longitude</td>
<td>float</td>
<td>Out</td>
<td>Optional</td>
<td>The longitude value of the address location.</td>
</tr>
<tr>
<td>latitude</td>
<td>float</td>
<td>Out</td>
<td>Optional</td>
<td>The latitude value of the address location.</td>
</tr>
<tr>
<td>resultcode</td>
<td>NVARCHAR(20)</td>
<td>Out</td>
<td>Optional</td>
<td>The result code of the address.</td>
</tr>
</tbody>
</table>

**Note:** The procedure reads the supplied parameters in order. If a parameter is required, it must not be omitted. It must either be set to NULL or left as an empty string if its type is string.

**Return Code Values and Results**

The Single_Fields procedure returns zero(0) on error or no candidate found and one (1) on success.
Example

Sample 1: Geocode a U.S. address with postal code.

DECLARE @ret AS INT
EXEC @ret = Single_Fields NULL, '48 RED COACH', 'SCHENECTADY, NY 12302', 'USA'
SELECT @ret AS N'GeocodeResult'
GO

The results might look like this in the database:

<table>
<thead>
<tr>
<th>Firm</th>
<th>Street</th>
<th>LastLine</th>
<th>Country</th>
<th>Longitude</th>
<th>Latitude</th>
<th>ResultCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>48 RED COACH DR</td>
<td>SCHENECTADY, NY 12302-3212 USA</td>
<td>-73.9655208738875</td>
<td>42.6457279894057</td>
<td>SSHPNTSCZA</td>
<td></td>
</tr>
</tbody>
</table>

Sample 2: Geocode a U.S. address with the default preference.

DECLARE @prefName AS NVARCHAR(30)
DECLARE @ret AS INT
EXEC @ret = Single_FieldsWithPrefName NULL, '1 main st', 'New York, NY', 'USA', @prefName OUTPUT
SELECT @ret AS N'GeocodeResult', @prefName AS N'PreferenceName'
GO

The results might look like this in the database:

<table>
<thead>
<tr>
<th>Firm</th>
<th>Street</th>
<th>LastLine</th>
<th>Country</th>
<th>Longitude</th>
<th>Latitude</th>
<th>ResultCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GOLDWATER MEMORIAL HOSPITAL</td>
<td>1 MAIN ST NEW YORK, NY 10044-0011 USA</td>
<td>-73.949</td>
<td>40.78303</td>
<td>52HFPNTSCZA</td>
<td></td>
</tr>
</tbody>
</table>

The results might look like this in the database:

More samples can be found in the demo directory after the Geocoding Extender is deployed.

Finding a Result Set

The second type of procedure available to geocode single addresses returns one or more candidates, when multiple candidates are available. These procedures are:

- Single_GeoAddr–Geocodes the given GeoAddress object.
- GeocodeAddress–Geocodes a single address and returns a result set of one or more candidates. Use when the input address information resides in separate fields.
- GeocodeAddressLastLine–Geocodes a single address and returns a result set of one or more candidates. Use when the input address information is given in one line in an unparsed form.
MapInfo Geocoding Extender for SQL Server does not determine the best candidate by dropping the other candidates, as the Single_Fields and FindLocation procedures do. These procedures are usually used when user interaction is necessary to determine the geocoding candidate or to allow the client application to have its own logic to decide which one is the best candidate. These procedures do not use tables for their input data; they get data in the form of parameters.

**Note:** You cannot use the trigger capabilities with these procedures. The trigger function needs a single row of results to operate, and these procedures return a result set.

### Single_GeoAddr

The Single_GeoAddr stored procedure geocodes the given GeoAddress object.

#### Syntax

```sql
CREATE PROCEDURE Single_GeoAddr
    @geoAddr GeoAddress,
    @retGeoAddr GeoAddress = NULL OUTPUT,
    @preference MM_Preference = NULL OUTPUT
AS
```

If the preference is in NVARCHAR, which indicates the preference name, the name of the stored procedure will be Single_GeoAddrWithPrefName.

```sql
CREATE PROCEDURE Single_GeoAddrWithPrefName
    @geoAddr GeoAddress,
    @retGeoAddr GeoAddress = NULL OUTPUT,
    @prefName NVARCHAR(30) = NULL OUTPUT
AS
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geoAddr</td>
<td>GeoAddress</td>
<td>In</td>
<td></td>
<td>The address to be geocoded.</td>
</tr>
<tr>
<td>retGeoAddr</td>
<td>GeoAddress</td>
<td>Out</td>
<td>Optional</td>
<td>The geocoded address.</td>
</tr>
</tbody>
</table>
Return Code Values and Results

The Single_GeoAddr stored procedure returns zero (0) on error, or the number of candidates found. It returns a GeoAddress object as the output parameter to carry the geocode results.

This procedure reads the supplied parameters in order. If a parameter is required, it must not be omitted. It must either be set to NULL or left as an empty string if its type is string.

Example

Sample 1: Geocode a U.S. address with preference name.

```sql
SET @inputAddress = CONVERT(GeoAddress, '')
SET @inputAddress.MainAddress = N'1 main st.'
SET @inputAddress.AreaName1 = N'NY'
SET @inputAddress.AreaName3 = N'New York'
SET @inputAddress.Country = N'USA'

DECLARE @mm_pref AS MM_Preference
SET @mm_pref = CONVERT(MM_Preference, '');
SET @mm_pref.CloseOnly = 'F'

DECLARE @numCandidates AS INT

EXEC @numCandidates = Single_GeoAddrWithPrefName @inputAddress, NULL, 'INITIALPREF'

SELECT @numCandidates AS N'NumberOfCandidates'
GO
```

The results might look like this:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefName</td>
<td>NVARCHAR(30)</td>
<td>In/Out</td>
<td>Optional</td>
<td>A user-defined preference name from the preference table. If it is missing or a null value, the default preference will be used.</td>
</tr>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>In/Out</td>
<td>Optional</td>
<td>An MM_Preference object. If it is missing or a null value, then the default preference as defined in the preference table will be used. If the preference table does not have a default preference then the default preference from the Geocoding Engine is used.</td>
</tr>
</tbody>
</table>
Sample 2: Geocode a U.S. address with default preference

DECLARE @inputAddress AS GeoAddress
DECLARE @geocodedAddress AS GeoAddress

SET @inputAddress = CONVERT(GeoAddress, '')
SET @inputAddress.MainAddress = N'1 main st.'
SET @inputAddress.AreaName1 = N'NY'
SET @inputAddress.AreaName3 = N'New York'
SET @inputAddress.Country = N'USA'

DECLARE @mm_pref AS MM_Preference
SET @mm_pref = CONVERT(MM_Preference, '');
SET @mm_pref.CloseOnly = 'F'

DECLARE @prefName AS NVARCHAR(30)
EXEC Single_GeoAddrWithPrefName @inputAddress, @geocodedAddress OUTPUT, @prefName OUTPUT

SELECT @geocodedAddress.GetNumCandidates() AS 'ResultCode', @prefName AS 'PreferenceName'
GO

The results might look like this:

<table>
<thead>
<tr>
<th>ResultCode</th>
<th>PreferenceName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INITIAL_PREF</td>
</tr>
</tbody>
</table>

More samples can be found in the demo directory after the Geocoding Extender is deployed.

**GeocodeAddress**

The GeocodeAddress procedure returns a result set as opposed to one row, as in the FindLocation procedure. The result set may consist of zero, one, or more rows. This procedure was called sp_geocodeAddress in the MapMarker ESP version of the product.

It is used if separate fields comprise the input data set. For example, firm, street, city, country division, postalcode, postalcode2, and preference are all passed as parameters for this procedure.

**Syntax**

CREATE PROCEDURE GeocodeAddress
    @firm NVARCHAR(256),
    @street NVARCHAR(256),
    @city NVARCHAR(40),
    @countrysub NVARCHAR(40),
    @postalcode NVARCHAR(10) = NULL,
@postalcode2 NVARCHAR(10) = NULL,
@country CHAR(3) = NULL,
@citysub CHAR(40) = NULL,
@preference MM_Preference = NULL OUTPUT,
@geocodedAddress GeoAddress = NULL OUTPUT
AS
...

If the preference is in NVARCHAR, which indicates the preference name, the name of the stored procedure will be GeocodeAddressWithPrefName.

CREATE PROCEDURE GeocodeAddressWithPrefName
    @firm NVARCHAR(256),
    @street NVARCHAR(256),
    @city NVARCHAR(40),
    @countrysub NVARCHAR(40),
    @postalcode NVARCHAR(16),
    @postalcode2 NVARCHAR(10) = NULL,
    @country CHAR(3) = NULL,
    @citysub CHAR(40) = NULL,
    @prefName NVARCHAR(30) = NULL OUTPUT,
    @geocodedAddress GeoAddress = NULL OUTPUT
AS
...

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>NVARCHAR(256),</td>
<td>In</td>
<td></td>
<td>The firm name.</td>
</tr>
<tr>
<td>street</td>
<td>NVARCHAR(256),</td>
<td>In</td>
<td></td>
<td>The street portion of the address.</td>
</tr>
<tr>
<td>city</td>
<td>NVARCHAR(40),</td>
<td>In</td>
<td></td>
<td>The city portion of the address.</td>
</tr>
<tr>
<td>countrysub</td>
<td>NVARCHAR(40),</td>
<td>In</td>
<td></td>
<td>The state or other country division portion of the address.</td>
</tr>
<tr>
<td>postalcode</td>
<td>NVARCHAR(16),</td>
<td>In</td>
<td>Optional</td>
<td>The postal code information for the address.</td>
</tr>
<tr>
<td>postalcode2</td>
<td>NVARCHAR(10),[]</td>
<td>In</td>
<td>Optional</td>
<td>The additional postal code information, such as the ZIP +4 extension in the United States.</td>
</tr>
</tbody>
</table>
Chapter 6: Geocoding a Single Address

Return Code Values and Result Set

This procedure returns zero (0) on error, or the number of candidates found.

This procedure reads the supplied parameters in order. If a parameter is required, it must not be omitted. It must either be set to NULL or left as an empty string if its type is string.

Example

Sample 1: Geocode a U.S. address with default preference settings.

```
DECLARE @numCandidates AS INT
DECLARE @geocodedAddress AS GeoAddress

EXEC @numCandidates = GeocodeAddress 'MapInfo', '', 'Troy', 'NY', NULL, NULL, 'USA', NULL, NULL, @geocodedAddress OUTPUT
SELECT @numCandidates AS N'NumberOfCandidates'
```

```
DECLARE @ctr AS INT
SET @ctr = 0; -- Initialize the counter variable
-- Loop through all the candidates in the geocoded address
WHILE (@ctr < @numCandidates)
BEGIN
    -- Get the candidate at given index
```
DECLARE @candRet Candidate
SET @candRet = @geocodedAddress.GetCandidate(@ctr)

--Print the results
SELECT
    @ctr AS Candidate,
    @candRet.FormattedStreet AS Street,
    @candRet.FormattedLocation AS LastLine,
    @candRet.X AS Longitude,
    @candRet.Y AS Latitude,
    @candRet.ResultCode AS ResultCode
--Increment the counter
SET @ctr = @ctr + 1
END
GO

The results might look like this:

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Street</th>
<th>LastLine</th>
<th>Longitude</th>
<th>Latitude</th>
<th>ResultCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GLOBAL W 1 TROY, NY 12180-0399</td>
<td>73.7029567812262</td>
<td>42.628937515421</td>
<td>S5HPNTSCZA</td>
<td></td>
</tr>
</tbody>
</table>

Sample 2: Geocode a U.S. address with default preference.

DECLARE @geocodedAddress AS GeoAddress
DECLARE @prefName AS NVARCHAR(30)
EXEC GeocodeAddressWithPrefName NULL, '1 main st', 'New York', 'NY', NULL, NULL, 'USA', NULL, @prefName OUTPUT, @geocodedAddress OUTPUT
SELECT @geocodedAddress.GetNumCandidates() AS 'CandidateCount', @prefName AS 'PreferenceName'
GO

The results might look like this:

<table>
<thead>
<tr>
<th>CandidateCount</th>
<th>PreferenceName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INITIALPREF</td>
</tr>
</tbody>
</table>

More samples can be found in the demo directory after the Geocoding Extender is deployed.

**GeocodeAddressLastLine**

The GeocodeAddressLastLine procedure returns a result as opposed to one row as in the Single_Fields procedure. The result set may consist of zero, one, or more rows. This procedure was called sp_geocodeAddressLastLine in the MapMarker ESP version of the product.

It is used if the input data is given as a single line containing the address information in an unparsed form. For example, firm, street, preference and LastLine (which includes city, country subdivision, postalcode, postaladdoncode and citysubdivision fields) are passed as parameters for in this procedure.
Syntax
CREATE PROCEDURE GeocodeAddressLastLine
    @firm NVARCHAR(256),
    @street NVARCHAR(256),
    @lastline NVARCHAR(256),
    @country CHAR(3) = NULL,
    @preference MM_Preference = NULL OUTPUT,
    @geocodedAddress GeoAddress = NULL OUTPUT
AS
...

If the preference is in NVARCHAR, which indicates the preference name, the name of the stored
procedure will be GeocodeAddressLastLineWithPrefName.

CREATE PROCEDURE GeocodeAddressLastLineWithPrefName
    @firm NVARCHAR(256),
    @street NVARCHAR(256),
    @lastline NVARCHAR(256),
    @country CHAR(3) = NULL,
    @prefName NVARCHAR(30) = NULL OUTPUT,
    @geocodedAddress GeoAddress = NULL OUTPUT
AS
...

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firm</td>
<td>NVARCHAR(256),</td>
<td>In</td>
<td></td>
<td>The firm name.</td>
</tr>
<tr>
<td>street</td>
<td>NVARCHAR(256),</td>
<td>In</td>
<td></td>
<td>The street portion of the address.</td>
</tr>
<tr>
<td>lastline</td>
<td>NVARCHAR(256),</td>
<td>In</td>
<td></td>
<td>String containing the unparsed city, state, postalcode and postaladdoncode information for the address.</td>
</tr>
<tr>
<td>country</td>
<td>CHAR(3)</td>
<td>In</td>
<td>Optional</td>
<td>The ISO three-letter country code of the address. If it is missing, the address will be considered to be in the default country, which is set up in the country table.</td>
</tr>
<tr>
<td>prefName</td>
<td>NVARCHAR(30)</td>
<td>In/Out</td>
<td>Optional</td>
<td>A user-defined preference name from the preference table. If it is missing or is a null value, the default preference will be used.</td>
</tr>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>In/Out</td>
<td>Optional</td>
<td>An MM_Preference object. If it is missing or is a null value, then the default preference as defined in the preference table will be used. If the preference table does not have a default preference then the default preference from the Geocoding Engine is used.</td>
</tr>
<tr>
<td>geocodeAddress</td>
<td>GeoAddress</td>
<td>Out</td>
<td>Optional</td>
<td>The geocoded address.</td>
</tr>
</tbody>
</table>

Note: This procedure reads the supplied parameters in order. If a parameter is required, it must not be omitted. It must either be set to NULL or left as an empty string if its type is string.

Return Code Values and Result Set

This procedure returns zero (0) on error, or the number of candidates found.

It returns a GeoAddress as the output parameter to carry the geocode results.
Example

Sample 1: Geocode to U.S. address with preference.

DECLARE @mm_pref AS MM_Preference
SET @mm_pref = CONVERT(MM_Preference, '');
SET @mm_pref.MaxCandidates = 10
DECLARE @geocodedAddress AS GeoAddress
DECLARE @numCandidates AS INT

EXEC @numCandidates = GeocodeAddressLastLine NULL, '1 main st', 'New York, NY', 'USA', @mm_pref OUTPUT, @geocodedAddress OUTPUT

DECLARE @ctr AS INT
SET @ctr = 0; -- Initialize the counter variable
-- Loop through all the candidates in the geocoded address
WHILE (@ctr < @numCandidates)
BEGIN
    -- Get the candidate at given index
    DECLARE @candRet Candidate
    SET @candRet = @geocodedAddress.GetCandidate(@ctr)

    -- Print the results
    SELECT
        @ctr AS Candidate,
        @candRet.FormattedStreet AS Street,
        @candRet.FormattedLocation AS LastLine,
        @candRet.X AS Longitude,
        @candRet.Y AS Latitude,
        @candRet.ResultCode AS ResultCode
    -- Increment the counter
    SET @ctr = @ctr + 1
END

GO

The results might look like this:
Sample 2: Geocode a U.S. address with default preference.

```sql
DECLARE @geocodedAddress AS GeoAddress
DECLARE @prefName AS NVARCHAR(30)

EXEC GeocodeAddressLastLineWithPrefName NULL, '1 main st', 'New York, NY', 'USA', @prefName OUTPUT, @geocodedAddress OUTPUT
SELECT @geocodedAddress.GetNumCandidates() AS 'CandidateCount', @prefName AS 'PreferenceName'
GO
```

More samples can be found in the demo directory after the Geocoding Extender is deployed.

Triggers

Triggers allow automatic single address geocoding. When an address is updated or added to a table, the trigger automatically geocodes the changes. The FindLocation and Single_Fields procedures geocode a single address and return the result into variables.

Use the FindLocation procedure if the input data set is divided into separate fields. For example, firm, street, city, countrysub, postalcode, postalcode2, country, citysub and preference are some of the parameters used with this procedure.

Use the Single_Fields procedure if the input data is given as a single line containing the address information in an unparsed form. For example, firm, street, preference and lastline (which includes city, countrysub, postalcode, postalcode2 and citysub fields) are some of the parameters used with this procedure.
Using Triggers

A database trigger is implemented by adding SQL code to a table on which you wish to perform automatic updates. You can use triggers in MapInfo Geocoding Extender for SQL Server to automatically update your table of data with new geocode results whenever you update or edit address records. Please refer to your SQL Server documentation for further information on what triggers are and how to use them.

Unlike the UDTs and stored procedures that are available in the Geocoding Extender, no predefined triggers exist after the Geocoding Extender is deployed.

The Geocoding Extender does, however, provide samples that show how to create database triggers based on the UDTs and stored procedures provided with the product. The sample triggers are for demonstration purposes only. They are created on test tables to illustrate their usage.

To create a sample trigger, you must run the `\demo\geocodeTrigger.sql` file.

The next sections provides some examples of how the sample triggers are implemented.

This sample trigger defines the geocoding functions that will be automatically executed when one or more columns are updated (including insertions).

**Example 1: Creating an Update Trigger**

The following script creates a trigger on the demo table, 'mmge_inout'. It automatically geocodes the new address when the address contents are updated.

```sql
CREATE TRIGGER geocode ON mmge_inout FOR UPDATE
AS
BEGIN
    DECLARE @firm NVARCHAR(256), @street NVARCHAR(256), @city NVARCHAR(256), @state NVARCHAR(256)
    DECLARE @zip NVARCHAR(10), @zip4 NVARCHAR(10)
    DECLARE @ozip NVARCHAR(10), @ozip4 NVARCHAR(10)
    DECLARE @lat float, @lon float, @result NVARCHAR(12)
    DECLARE @prefName NVARCHAR(30)
    DECLARE @prim NVARCHAR(50)

    DECLARE ins CURSOR LOCAL FOR SELECT prim, firm, adr, city, state, zip, zip4 FROM inserted
    BEGIN
        SET NOCOUNT ON
        SET @prefName = N'INITIALPREF'
        IF UPDATE(adr) OR UPDATE(firm) OR UPDATE(city) OR UPDATE(state) OR UPDATE(zip) OR UPDATE(zip4)
        BEGIN
            IF cursor_status('LOCAL','ins') <> -1 CLOSE ins
            OPEN ins
```
Example 2: Creating an Insert Trigger

The following script creates a trigger on the demo table, ‘mmge_inout’. It automatically geocodes the new address when a new record is added to the table.

CREATE TRIGGER geocodeIns ON mmge_inout FOR INSERT AS
BEGIN
    DECLARE @firm NVARCHAR(256), @street NVARCHAR(256), @city NVARCHAR(40),
    @state NVARCHAR(40)
    DECLARE @zip NVARCHAR(10), @zip4 NVARCHAR(10)
    DECLARE @prefName NVARCHAR(30)
    DECLARE @prim NVARCHAR(50)

    DECLARE ins CURSOR LOCAL  FOR SELECT prim, firm, adr, city, state, zip,
    zip4 FROM inserted
    BEGIN
        SET NOCOUNT ON
        FETCH NEXT FROM ins INTO @prim, @firm, @street, @city, @state, @zip, @zip4
        WHILE (@@FETCH_STATUS = 0)
            BEGIN
                EXEC FindLocationWithPrefName @firm, @street, @city,
                @state, @zip, @zip4, 'USA', NULL, @prefName, @lon OUTPUT, @lat OUTPUT,
                @result OUTPUT
                UPDATE mmge_inout SET ofirm=@firm , oadr=@street, ocity=@city,
                ostate=@state, ozip=@zip, ozip4=@zip4, olat=@lat, olong=@lon,
                result=@result WHERE prim = @prim
                FETCH NEXT FROM ins INTO @prim, @firm, @street, @city, @state, @zip, @zip4
            END
        CLOSE ins
    END
    SET NOCOUNT OFF
END
GO

-- Test the trigger
PRINT 'Update Test'
UPDATE mmge_inout SET firm=NULL WHERE prim = 'mapinfo'
SELECT * FROM mmge_inout WHERE prim = 'mapinfo'
GO
SET @prefName = N'INITIALPREF'
IF cursor_status('LOCAL','ins') <> -1 CLOSE ins
OPEN ins
FETCH NEXT FROM ins INTO @prim,@firm, @street, @city, @state, @zip, @zip4
WHILE (@@FETCH_STATUS = 0)
BEGIN
    UPDATE mmge_inout SET firm=@firm WHERE prim = @prim
    FETCH NEXT FROM ins INTO @prim,@firm, @street, @city, @state,@zip, @zip4
    END
CLOSE ins
SET NOCOUNT OFF
END
END
GO

-- Test the trigger
PRINT 'Insert test'
INSERT INTO mmge_inout (prim,firm,adr,city,state) VALUES ('mapinfo','mapinfo','1 global view','troy','ny');
SELECT * FROM mmge_inout WHERE prim = 'mapinfo'
GO
Preference Settings

Preferences determine the conditions under which an address record is geocoded and what kind of information is returned in the output.

It is important to keep in mind how you intend to use your geocoded data when you set preferences. Preference settings can affect how many addresses in a table are successfully geocoded and the locational accuracy of the geocoded point on a map. This chapter describes the preference settings in MapInfo Geocoding Extender for SQL Server and explains how to manage them.

In this chapter:

- Using Preferences ..................................................88
- Managing Geocoding Preferences .........................91
- Using Country Tables ............................................99
- Managing Country Table Settings .........................100
Using Preferences

A preference is used to define how a geocode request is handled. It contains several different settings options. This enables you to define the settings you require for your geocoding.

Definitions for geocoding preferences in MapInfo Geocoding Extender for SQL Server are stored as records in a table called mmge_preferences. Only one of them is marked as default. This default preference is set to use the same defaults as MapMarker.

When geocoding, you can specify the preference content in the following ways:

- Use a preference name, which refers to a record in the mmge_preferences table.
- Use an MM_Preference object.

When adding a new preference item to the mmge_preferences table, the MM_Preference object is used to carry the new preference content.

These settings can be modified for your own needs. Several procedures have been supplied to enable you to edit preferences. Whenever a preference is not specified by a particular procedure, the default preference is used. For example, for stored procedures that take an MM_Preference object as one of the parameters, if a NULL value is given the Geocoding Extender will use the default preference stored in the mmge_preferences table. Otherwise, the Geocoding Extender will take the preference content stored in the MM_Preference object.

Preferences Table

The mmge_preferences table provides initial default settings for geocoding preferences. The default settings are in a preference called 'INITIALPREF'. This is the initial default preference name. The default values in the mmge_preferences table are the values given in "INITIALPREF". If you wish to change any of the settings in the preferences, you must do so before you start to geocode.

The following gives a description of all the columns in the preferences table:
### Chapter 7: Preference Settings

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>NNVARCHAR(30)</td>
<td>A unique identifier or name for the preferences.</td>
</tr>
<tr>
<td>COUNTRYTABLE</td>
<td>NNVARCHAR(20)</td>
<td>This parameter should be set to the country table name. The default name is mmge_countries.</td>
</tr>
<tr>
<td>MUSTMATCHADDRESSNUMBER</td>
<td>char(1)</td>
<td>T to require exact match on house number. T=True, F=False. Default is T.</td>
</tr>
<tr>
<td>MUSTMATCHMAINADDRESS</td>
<td>char(1)</td>
<td>T to require exact match on street name. T=True, F=False. Default is F.</td>
</tr>
<tr>
<td>MUSTMATCHPOSTAL</td>
<td>char(1)</td>
<td>T to require exact match on postal code. T=True, F=False. Default is F.</td>
</tr>
<tr>
<td>MUSTMATCHINPUT</td>
<td>char(1)</td>
<td>T to require exact match on all the input address fields. T=True, F=False. Default is F.</td>
</tr>
<tr>
<td>MUSTMATCHAREANAME1</td>
<td>char(1)</td>
<td>T to require exact match on area name1. T=True, F=False. Default is T.</td>
</tr>
<tr>
<td>MUSTMATCHAREANAME2</td>
<td>char(1)</td>
<td>T to require exact match on area name 2. T=True, F=False. Default is T.</td>
</tr>
<tr>
<td>MUSTMATCHAREANAME3</td>
<td>char(1)</td>
<td>T to require exact match on area name 3. T=True, F=False. Default is T.</td>
</tr>
<tr>
<td>MUSTMATCHAREANAME4</td>
<td>char(1)</td>
<td>T to require exact match on area name 4. T=True, F=False. Default is T.</td>
</tr>
<tr>
<td>MAXCANDIDATES</td>
<td>int</td>
<td>The maximum number of candidates to return for a geocoding request. Default is 5.</td>
</tr>
<tr>
<td>MAXRANGESPERCANDIDATE</td>
<td>int</td>
<td>Maximum number of AddressRange per Candidate.</td>
</tr>
<tr>
<td>MAXRANGEUNITSPPERRANGE</td>
<td>int</td>
<td>Maximum number of AddressRangeUnit per AddressRange.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>POSTALFALLBACK</td>
<td>char(1)</td>
<td>Fall back to postal code centroid if street level address cannot be geocoded.</td>
</tr>
<tr>
<td>GEOGRAPHICFALLBACK</td>
<td>char(1)</td>
<td>Fall back to geographic area centroid if street level address cannot be geocoded.</td>
</tr>
<tr>
<td>CORNEROFFSETUNIT</td>
<td>char(2)</td>
<td>The distance unit of the corner offset.</td>
</tr>
<tr>
<td>CORNEROFFSETDISTANCE</td>
<td>number</td>
<td>The value of the street offset distance.</td>
</tr>
<tr>
<td>STREETOFFSETUNIT</td>
<td>char(2)</td>
<td>The distance unit of the street offset.</td>
</tr>
<tr>
<td>STREETOFFSETDISTANCE</td>
<td>number</td>
<td>The value of the street offset distance.</td>
</tr>
<tr>
<td>REQUESTSRID</td>
<td>NNVARCHAR(30)</td>
<td>The coordinate reference system name to be used in the geocoding result. (DEFAULT: EPSG: 4326, (WGS-84)).</td>
</tr>
<tr>
<td>CLOSEONLY</td>
<td>char(1)</td>
<td>T indicates that only close match candidates will be returned. F means return all candidates. T=True, F=False. Default is T.</td>
</tr>
<tr>
<td>BATCHSIZE</td>
<td>int</td>
<td>The number of grouped geocode requests returned in one batch when geocoding a table. Since returned information can be lengthy, it is recommended that your batch size is below 20. This is a tuning parameter and will not affect the geocoding itself.</td>
</tr>
<tr>
<td>DICTIONARYPREFERENCE</td>
<td>NNVARCHAR(256)</td>
<td>The preference of data dictionaries.</td>
</tr>
<tr>
<td>DEFAULTPREFERENCE</td>
<td>char(1)</td>
<td>T indicates that the current preference is set as the default. T=True, F=False. Default is F.</td>
</tr>
</tbody>
</table>
### Managing Geocoding Preferences

MapInfo Geocoding Extender for SQL Server preferences are initially set to use the same defaults as MapMarker, including a required exact match to house number and a relaxed match to street name and postal code. If you wish, you can leave the default settings as they are. However, MapMarker servers in different countries may have different default preference settings. Users can define different preference records in the preference table to match different requirements.

The Geocoding Extender enables you to change the default settings. You can create, set, and delete preferences, set default preferences, and determine which preference is currently set as the default.

There are six procedures supplied with MapInfo Geocoding Extender for SQL Server that enable you to perform these tasks. They are:

- `create_preference`—create a new preference.
- `delete_preference`—delete an existing preference.
- `set_default_preference`—set a preference as the default preference.
- `get_default_preference`—print the default preference name on the screen.
- `set_preference`—modify the content of a preference in the preference table.
- `get_preference`—print the specified preference and its setting.

The sections that follow describe each one.

### Creating a Preference

The `create_preference` procedure creates a new preference and saves it in the `mmge_preferences` table. See [Preferences Table on page 88](#) for a description of the preference settings.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGFILE</td>
<td>NVARCHAR(256)</td>
<td>The full path name of the log file. It applies to table geocoding only. If it is set, all the warning and error messages will be put in the log file rather than the SQL Server management studio console.</td>
</tr>
<tr>
<td>LOGLEVEL</td>
<td>int</td>
<td>The log level. It applies to table geocoding only. The possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• E = Log errors only;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• W = Log errors and warning messages</td>
</tr>
</tbody>
</table>

Note: If no preference is specified as the default, MapInfo Geocoding Extender for SQL Server raises an exception to notify the user.
Managing Geocoding Preferences

Syntax
CREATE PROCEDURE create_preference
    @preference MM_Preference
AS
...

Parameters
The parameters this procedure takes are given below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td></td>
<td></td>
<td>The new preference to be created. The MM_Preference UDT object holds the preference content.</td>
</tr>
</tbody>
</table>

Return Code Values and Results
The create_preference procedure creates a preference record in the mmge_preferences table with the given preference settings.

It returns zero (0) on error and one (1) on success.

For example, if the preference name in the input MM_Preference exists in the preference table already, an error message will be printed and the return value will be 0.

Example
Sample 1: Create a preference setting with the given name in the mmge_preferences table. By default, the newly created preference is not the default preference. Use the set_default_preference procedure to set the preference you just created as the default preference.

DECLARE @preference as MM_Preference

SET @preference = CONVERT(MM_Preference, '')

SET @preference.Name = 'MyPreference'
SET @preference.CountryTable = 'mmge_countries'
SET @preference.CloseOnly = 'F'
SET @preference.MaxCandidates = '-1'
SET @preference.MaxRangesPerCandidate = '10'
SET @preference.MaxRangeUnitsPerRange = '10'
SET @preference.BatchSize = 100
SET @preference.PostalFallback = 'T'
SET @preference.GeographicFallback = 'T'
SET @preference.CornerOffsetUnit = 'm'
SET @preference.CornerOffsetDistance = 10
Chapter 7: Preference Settings

SET @preference.StreetOffsetUnit = 'm'
SET @preference.StreetOffsetDistance = 10
SET @preference.RequestSRID = 'epsg:4326'
SET @preference.MustMatchAddressNumber = 'F'
SET @preference.MustMatchMainAddress = 'T'
SET @preference.MustMatchAreaName1 = 'T'
SET @preference.MustMatchAreaName2 = 'F'
SET @preference.MustMatchAreaName3 = 'T'
SET @preference.MustMatchAreaName4 = 'F'
SET @preference.MustMatchPostal = 'F'
SET @preference.MustMatchInput = 'F'
SET @preference.DictionaryPreference = 'AD_AND_UD'

DECLARE @ret as INT
EXEC @ret = create_preference @preference
SELECT @ret as ReturnValue
GO

Deleting a Preference

The delete_preference procedure deletes an existing preference. If the default preference is deleted, a warning message will be printed. You invoke the procedure by supplying the preference name to be deleted.

Syntax

CREATE PROCEDURE delete_preference
    @preferenceName NVARCHAR(30);
AS
...

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferenceName</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td></td>
<td>The name of the preference to be deleted from the mmge_preferences table.</td>
</tr>
</tbody>
</table>

Return Code Values and Results

The delete_preference procedure removes the preference record with the given preference name from the mmge_preferences table.
Managing Geocoding Preferences

It returns zero (0) on error or one (1) on success.

Example

Sample 1: Delete a preference setting with the given name in the mmge_preferences table.

DECLARE @ret as INT
EXEC @ret = delete_preference 'MyPreference'
SELECT @ret as ReturnValue
GO

Assigning a Default Preference

The set_default_preference procedure assigns a preference as a default preference. The corresponding default_preference entry in the mmge_preferences table is changed to 'T' and all the others in that column are changed to 'F'.

Syntax

CREATE PROCEDURE set_default_preference
    @preferenceName NVARCHAR(30)
AS
...

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferenceName</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td></td>
<td>The name of the preference to be set as the default in the mmge_preferences table.</td>
</tr>
</tbody>
</table>

Return Code Values and Results

The set_default_preference procedure sets the given preference settings as the default preference. The default preference will be used when no preference name is provided when geocoding (NULL value for the preference name or MM_Preference object).

It returns zero (0) on error or one (1) on success.

Example

Sample 1: Set the given preference name as the default preference.

DECLARE @ret as INT
EXEC @ret = set_default_preference 'MyPreference'
SELECT @ret as ReturnValue
GO

<table>
<thead>
<tr>
<th>NAME</th>
<th>INITIALPREF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Getting the Default Preference

The `get_default_preference` procedure retrieves the content of the default preference.

#### Syntax

```sql
CREATE PROCEDURE [dbo]. [get_default_preference]
    @preferenceName NVARCHAR(30) = NULL OUTPUT,
    @retPreference MM_Preference = NULL OUTPUT
AS
...
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferenceName</td>
<td>NVARCHAR(30)</td>
<td>Output</td>
<td>Optional</td>
<td>The name of the default preference in the mmge_preferences table.</td>
</tr>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>Output</td>
<td>Optional</td>
<td>The MM_Preference object that holds the content of the default preference settings.</td>
</tr>
</tbody>
</table>

#### Return Code Values and Results

The `get_default_preference` procedure returns the name and the contents of the default preference. It returns zero (0) on error or one (1) on success.

#### Example

Sample 1: Get the default preference name.

```sql
DECLARE @prefName as NVARCHAR(20)
EXEC get_default_preference @prefName OUTPUT
SELECT @prefName as DefaultPreferenceName
GO
```

<table>
<thead>
<tr>
<th>DefaultPreferenceName</th>
<th>INITIALPREF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Sample 2: Get the default preference name and UDT object.

DECLARE @prefName as NVARCHAR(20)
DECLARE @preference as MM_Preference
EXEC get_default_preference @prefName OUTPUT, @preference OUTPUT
SELECT @preference.Name as DefaultPreferenceName
GO

Setting a Preference

The set_preference procedure allows you to change the settings of a particular preference.

Note: You cannot use this stored procedure to specify whether the preference is the default preference. You must perform a subsequent call using the set_default_preference procedure to set a preference you have modified as the default.

Syntax

CREATE PROCEDURE set_preference
    @preferenceName MM_Preference
AS
...

Parameters

The parameters this procedure takes are given below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>In</td>
<td></td>
<td>The MM_Preference object that holds the contents of the specified preference.</td>
</tr>
</tbody>
</table>

Return Code Values and Results

The set_preference procedure modifies a specific preference setting with new values.

If the given preference does not exist in the preference table, the Geocoding Extender will create a new one with the given content.

This procedure returns zero (0) on error or one (1) on success.

Example

Sample 1: Update a preference settings in the mmge_preferences table.

DECLARE @preference as MM_Preference

SET @preference = CONVERT(MM_Preference, '')
SET @preference.Name = 'MyPreference'
SET @preference.CountryTable = 'mmge_countries'
SET @preference.CloseOnly = 'F'
SET @preference.MaxCandidates = '-1'
SET @preference.MaxRangesPerCandidate = '10'
SET @preference.MaxRangeUnitsPerRange = '10'
SET @preference.BatchSize = 100
SET @preference.PostalFallback = 'T'
SET @preference.GeographicFallback = 'T'
SET @preference.CornerOffsetUnit = 'm'
SET @preference.CornerOffsetDistance = 10
SET @preference.StreetOffsetUnit = 'm'
SET @preference.StreetOffsetDistance = 10
SET @preference.RequestSRID = 'epsg:4326'
SET @preference.MustMatchAddressNumber = 'F'
SET @preference.MustMatchMainAddress = 'T'
SET @preference.MustMatchAreaName1 = 'T'
SET @preference.MustMatchAreaName2 = 'F'
SET @preference.MustMatchAreaName3 = 'T'
SET @preference.MustMatchAreaName4 = 'F'
SET @preference.MustMatchPostal = 'F'
SET @preference.MustMatchInput = 'F'
SET @preference.DictionaryPreference = 'AD_AND_UD'

DECLARE @ret as INT
EXEC @ret = set_preference @preference
SELECT @ret as ReturnValue
GO

### Getting Preference Settings

The `get_preference` procedure returns all the current settings of a particular preference.

**Syntax**

```
get_preference @preferenceName;
```
Managing Geocoding Preferences

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferenceName</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The name of the preference in the mmge_preferences table.</td>
</tr>
<tr>
<td>preference</td>
<td>MM_Preference</td>
<td>Output</td>
<td>Optional</td>
<td>The MM_Preference object that holds the contents of the specified preference.</td>
</tr>
</tbody>
</table>

Return Code Values and Results

This procedure gets the preference in the output MM_Preference object.

It returns zero (0) on error or one (1) on success.

Example

This example retrieves the settings for “preferenceName1”.

Sample 1: Read the preference settings with the given name into a MM_Preference UDT object.

```sql
DECLARE @preference as MM_Preference
EXEC get_preference 'MyPreference', @preference OUTPUT

SELECT
    @preference.Name AS Name,
    @preference.CountryTable AS CountryTable,
    @preference.CloseOnly AS CloseMatchesOnly,
    @preference.MaxCandidates AS MaxCandidatesReturned,
    @preference.MaxRangesPerCandidate AS MaxRangesPerCandidate,
    @preference.MaxRangeUnitsPerRange AS MaxRangeUnitPerRange,
    @preference.BatchSize AS BatchSize,
    @preference.PostalFallback AS FallbackToPostal,
    @preference.GeographicFallback AS FallbackToGeographic,
    @preference.CornerOffsetUnit AS CornerOffsetUnit,
    @preference.CornerOffsetDistance AS CornerOffsetDistance,
    @preference.StreetOffsetUnit AS StreetOffsetUnit,
    @preference.StreetOffsetDistance AS StreetOffsetDistance,
    @preference.RequestSRID AS RequestSRID,
    @preference.MustMatchAddressNumber AS MustMatchAddressNumber,
    @preference.MustMatchMainAddress AS MustMatchMainAddress,
    @preference.MustMatchAreaName1 AS MustMatchAreaName1,
    @preference.MustMatchAreaName2 AS MustMatchAreaName2,
    @preference.MustMatchAreaName3 AS MustMatchAreaName3,
    @preference.MustMatchAreaName4 AS MustMatchAreaName4,
    @preference.MustMatchPostal AS MustMatchPostal,
```
Using Country Tables

The country table is one of the preference settings in the mmge_preferences table. This preference indicates the name of the country table to use when you geocode. The default country table name in the preferences table is mmge_countries.

The country table supplies geocoding server information about each country that MapInfo Geocoding Extender for SQL Server looks for when it geocodes. You can add or remove countries from the table as needed. Multiple country tables can also be set up to carry different country and server URL mappings.

The Geocoding Extender can talk to multiple MapMarker or Envinsa server instances based on the country of the input address. Each country has one server URL of the geocoding server associated with it. This geocoding server can be either MapMarker Java server or the Envinsa Location Utility Service.

The following gives a description of the columns in the country table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRYID</td>
<td>char(3)</td>
<td>An ISO three-letter code to represent a country or area, e.g., USA.</td>
</tr>
<tr>
<td>SERVERURL</td>
<td>NVARCHAR(150)</td>
<td>The URL of the geocoding server that is able to geocode addresses in this country.</td>
</tr>
<tr>
<td>SERVERTYPE</td>
<td>NVARCHAR(10)</td>
<td>The type of the geocoding server. It has two possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Envinsa–The server is an Envinsa geocoding service instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MapMarker–The server is a MapMarker Plus instance. This is the default type.</td>
</tr>
<tr>
<td>ISDEFAULT</td>
<td>char(1)</td>
<td>Indicates whether this country is the default. T is default. F is not. (Optional) Default is F.</td>
</tr>
<tr>
<td>USERNAME</td>
<td>NVARCHAR(10)</td>
<td>The user name used to connect to the server. This is valid for Envinsa servers only.</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>NVARCHAR(10)</td>
<td>The password used with the user name to connect to the server. This is valid for Envinsa servers only.</td>
</tr>
</tbody>
</table>
Please take note of the following as you use country tables:

- If the input address does not have the country code filled in, the Geocoding Extender will use the default country code and server/URL. If using InputTableInfo, the DefaultCountry property takes higher priority. If it is not set, then use this default country in the country table.
- If the input address does have a country code, but it is not listed in the country table, the Geocoding Extender will use the server/URL of the default country as the geocoding engine.
- There can be only one default country in a country table.

Managing Country Table Settings

Stored procedures included with MapInfo Geocoding Extender for SQL Server enable you to set up and manage country tables. They are:

- add_country—add a new country to specified country table.
- delete_country—delete an existing country from specified country table.
- get_country—returns server URL and server type of the specified country.
- set_country—sets the country/server URL association in the specified country table.
- get_product_version—returns the version and type of the geocode server used for the given country.

The sections that follow describe each one.

Adding a Country

The add_country procedure enables you to add a country/server URL pair to a specified country table. When you reference that country table in the geocoding preferences, the Geocoding Extender can geocode addresses to the country that you added to that country table.

Syntax

```
CREATE PROCEDURE add_country
    @countryID NVARCHAR(3),
    @serverURL NVARCHAR(150),
    @serverType NVARCHAR(10),
    @isDefault CHAR(1) = 'F',
    @username NVARCHAR(30) = NULL,
    @password NVARCHAR(30) = NULL,
    @countryTable NVARCHAR(30) = N'mmge_countries'
AS
    ...
```

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>countryID</td>
<td>CHAR(3)</td>
<td>In</td>
<td></td>
<td>The ISO three-letter country code.</td>
</tr>
<tr>
<td>serverURL</td>
<td>NVARCHAR(150)</td>
<td>In</td>
<td></td>
<td>The URL of the geocoding server.</td>
</tr>
<tr>
<td>serverType</td>
<td>NVARCHAR(10)</td>
<td>In</td>
<td></td>
<td>The type of the geocoding server. It can be either 'MapMarker' or 'Envinsa'.</td>
</tr>
<tr>
<td>isDefault</td>
<td>CHAR(1)</td>
<td>In</td>
<td></td>
<td><strong>T</strong> = Is default country. <strong>F</strong> = Is not default country (default).</td>
</tr>
<tr>
<td>username</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The user name used in the Envinsa server. It only applies when the serverType is Envinsa.</td>
</tr>
<tr>
<td>password</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The password used in the Envinsa server. It only applies when the serverType is Envinsa.</td>
</tr>
<tr>
<td>countryTable</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The name of the country table where the country is to be added. The default value is 'mmge_countries'.</td>
</tr>
</tbody>
</table>

Return Code Values and Results

This procedure returns zero (0) on error or one (1) on success.

Example

Sample 1: Add a country using MapMarker as the geocode engine.

```
DECLARE @ret as INT
EXEC @ret = add_country 'USA', 'http://localhost:8095/mapmarker40/servlet/mapmarker', 'MapMarker'
SELECT @ret as ReturnValue
GO
```

Sample 2: Add a default country using Envinsa as the geocoding engine. Any address that does not have a country code registered in the country table will use this default setting.
Managing Country Table Settings

DECLARE @ret as INT
EXEC @ret = add_country 'CAN', 'http://localhost:1980/LocationUtility/services/LocationUtility', 'Envinsa', 'T', 'demo', 'demo', 'mmge_countries'
SELECT @ret as ReturnValue
GO

## Deleting a Country

This procedure enables you to delete a country/server URL pair from a specified country table.

### Syntax

```sql
CREATE PROCEDURE delete_country
    @countryID CHAR(3),
    @countryTable NVARCHAR(30) = N'mmge_countries'
```

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>countryID</td>
<td>CHAR(3)</td>
<td>In</td>
<td>Optional</td>
<td>The ISO three-letter country code.</td>
</tr>
<tr>
<td>countryTable</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The name of the country table from which the specified country/server URL pair is to be deleted</td>
</tr>
</tbody>
</table>

### Return Code Values and Results

This procedure returns zero (0) on error or one (1) on success.

### Example

Sample 1: Delete a country with a given country table name and an ISO three-letter country code.

DECLARE @ret as INT
EXEC @ret = delete_country 'USA', 'mmge_countries'
SELECT @ret as ReturnValue
GO
Getting a Country

The get_country procedure returns the server URL, server type, and other information for the specified country.

Syntax

CREATE PROCEDURE get_country
    @countryID NVARCHAR(3),
    @serverURL NVARCHAR(150) = NULL OUTPUT,
    @serverType NVARCHAR(10) = NULL OUTPUT,
    @isDefault CHAR(1) = NULL OUTPUT,
    @username NVARCHAR(30) = NULL OUTPUT,
    @password NVARCHAR(30) = NULL OUTPUT,
    @countryTable NVARCHAR(30) = N'mmge_countries'

AS

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>countryID</td>
<td>CHAR(3)</td>
<td>In</td>
<td></td>
<td>The ISO three-letter country code.</td>
</tr>
<tr>
<td>serverURL</td>
<td>NVARCHAR(150)</td>
<td>Out</td>
<td>Optional</td>
<td>The URL of the geocoding server.</td>
</tr>
<tr>
<td>serverType</td>
<td>NVARCHAR(10)</td>
<td>Out</td>
<td>Optional</td>
<td>The type of the geocoding server. It can be either 'MapMarker' or 'Envinsa'.</td>
</tr>
<tr>
<td>isDefault</td>
<td>char(1)</td>
<td>Out</td>
<td>Optional</td>
<td>T = Is default country. F = Is not default country (default).</td>
</tr>
<tr>
<td>username</td>
<td>NVARCHAR(30)</td>
<td>Out</td>
<td>Optional</td>
<td>The user name used in the Envinsa server. It only applies when the serverType is Envinsa.</td>
</tr>
<tr>
<td>password</td>
<td>NVARCHAR(30)</td>
<td>Out</td>
<td>Optional</td>
<td>The password used in the Envinsa server. It only applies when the serverType is Envinsa.</td>
</tr>
<tr>
<td>countryTable</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The name of the country table.</td>
</tr>
</tbody>
</table>

Return Code Values and Results

This procedure returns zero (0) on error or one (1) on success.
Example

Sample 1: Get the server URL for a given ISO three-letter country code in a country table.

DECLARE @ret AS INT
DECLARE @serverURL as NVARCHAR(100)
EXEC @ret = get_country 'USA', @serverURL OUTPUT
SELECT @serverURL as ServerURL
SELECT @ret AS ReturnValue
GO

Sample 2: Get the server information for a given ISO three-letter country code in a country table.

DECLARE @ret AS INT
DECLARE @serverURL AS NVARCHAR(100)
DECLARE @serverType AS NVARCHAR(10)
DECLARE @isDefault AS CHAR(1)
DECLARE @username AS NVARCHAR(30)
DECLARE @password AS NVARCHAR(30)
EXEC @ret = get_country 'CAN', @serverURL OUTPUT, @serverType OUTPUT, @isDefault OUTPUT, @username OUTPUT, @password OUTPUT, 'mmge_countries'
SELECT @serverURL as ServerURL, @serverType as ServerType, @isDefault as IsDefault, @username as UserName, @password as Password, @ret AS ReturnValue
GO

Setting the Country

The set_country procedure sets the country/server URL association and other information in the specified country table.

Syntax

CREATE PROCEDURE [dbo].[set_country]
    @countryID NVARCHAR(3),
    @serverURL NVARCHAR(150),
Chapter 7: Preference Settings

@serverType NVARCHAR(10),
@isDefault CHAR(1),
@username NVARCHAR(30) = NULL,
@password NVARCHAR(30) = NULL,
@countryTable NVARCHAR(30) = N'mmge_countries'
AS

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>countryID</td>
<td>CHAR(3)</td>
<td>In</td>
<td></td>
<td>The ISO three-letter country code.</td>
</tr>
<tr>
<td>serverURL</td>
<td>NVARCHAR(150)</td>
<td>In</td>
<td></td>
<td>The URL of the geocoding server.</td>
</tr>
<tr>
<td>serverType</td>
<td>NVARCHAR(10)</td>
<td>In</td>
<td></td>
<td>The type of the geocoding server. It can be either ‘MapMarker’ or ‘Envinsa’.</td>
</tr>
<tr>
<td>isDefault</td>
<td>char(1)</td>
<td>In</td>
<td></td>
<td>T = Is default country.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F = Is not default country (default).</td>
</tr>
<tr>
<td>username</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The user name used in the Envinsa server. It only applies when the serverType is Envinsa.</td>
</tr>
<tr>
<td>password</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The password used in the Envinsa server. It only applies when the serverType is Envinsa.</td>
</tr>
<tr>
<td>countryTable</td>
<td>NVARCHAR(30)</td>
<td>In</td>
<td>Optional</td>
<td>The name of the country table. The default value is ‘mmge_countries’.</td>
</tr>
</tbody>
</table>

Return Code Values and Results

This procedure returns zero (0) on error or and one (1) on success. If the given country does not exist in the country table, a new one with the given content will be added in the country table.

Examples

Sample 1: Set the server URL, server type, is default server flag for a given ISO three-letter country code in a country table.

DECLARE @ret AS INT
Managing Country Table Settings

EXEC @ret = set_country 'USA', 'http://localhost:8095/mapmarker40/servlet/mapmarker', 'MapMarker', 'F'
SELECT @ret AS ReturnValue
GO

Sample 2: Set the server URL, server type, is default server flag, username, and password for a given ISO three-letter country code in a country table.

DECLARE @ret AS INT
EXEC @ret = set_country 'CAN', 'http://localhost:1980/LocationUtility/services/LocationUtility', 'Envinsa', 'T', 'demo', 'demo', 'mmge_countries'
SELECT @ret AS ReturnValue
GO

Getting the Server Version

The get_product_version procedure returns the version and type of the geocoding server used for the given country.

Syntax
CREATE PROCEDURE get_product_version
  @country NVARCHAR(3),
  @sResult NVARCHAR(1024) = NULL OUTPUT,
  @countryTable NVARCHAR(30) = N'mmge_countries'
AS
  ...

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>In/Out</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>country</td>
<td>CHAR(3)</td>
<td>In</td>
<td></td>
<td>The ISO three-letter country code.</td>
</tr>
</tbody>
</table>
Return Code Values and Results

Prints both the server types, MapMarker or Envinsa, and the server version information.

Example

Sample 1: Get the geocoding server version for USA.

EXEC get_product_version 'USA'
GO

Sample 2: Get the geocoding server version for Canada.

EXEC get_product_version 'CAN'
GO
This chapter explains how to read the result codes that the MapMarker and Envinsa server return after a geocoding operation.
Result Codes

MapMarker and Envinsa return a result code for every address they attempt to match. The code represents whether a match was made, the type of match it is, and gives information about the quality of the match.

The result code is an alphanumeric code of 1-10 characters. They fall into three categories:

- Single close match to street level (S category)
- Postal Code centroid match (Z category)
- Non-matches (N category)

For S and Z categories, the first two characters represent the positional accuracy of the match, for example, where the point for the record would spot on a street map.

For the S category matches there are an additional eight characters that define the precision with which the individual components in the input address were matched. If there was no match on a particular address component, the code would contain a dash. For example, a single close match to a street address that matched to all address components except house number would look like: S5-PNTSCZA.

The result codes that MapMarker and Envinsa use are based on the result codes for United States addresses. Although the codes were originally designed for the United States, the same codes are used by all the countries that MapMarker and Envinsa support. Because of this, when you geocode addresses from other countries there may not be an exact correspondence between the parts of the address represented by the characters in the result code and the address elements used in the particular country.
# Chapter 8: Result Codes Defined

## S Category: Single Close Match to Street Level

<table>
<thead>
<tr>
<th>Code</th>
<th>Match Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S8</td>
<td>Matched to a street address, point located at either the single point associated with an address point candidate or at an address point candidate that shares the same house number.</td>
</tr>
</tbody>
</table>
| S7   | Matched to a street address, point located at an interpolated point along the candidate’s street segment. This result is returned using address point interpolation. When the candidate to be returned is not an address point candidate and there are no exact matches on house number among other address point candidates generated in the candidate list, all address point candidates are filtered according to the following criteria:  
  - 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, that is, one of the candidate’s ranges must intersect the address point candidate’s house number.  

  When the filtering is complete, the resulting list of address point candidates is used to perform the interpolation. The point is interpolated according to the next highest or lowest address point candidate that both intersects the segment and whose house number is contained within the range of houses of the original candidate. |
| S6   | Matched to a location, point located at point ZIP™ centroid (U.S. only)            |
| S5   | Matched to a street address.                                                      |
| S4   | Matched to an interpolated point on the street segment.                           |
| S3   | Matched to a ZIP + 4 centroid (centerpoint of the ZIP + 4 boundary) (U.S. only).   |
| S2   | Matched to a ZIP + 2 centroid (centerpoint of the ZIP + 2 boundary) (U.S. only).   |
| S1   | Matched to a ZIP Code centroid (centerpoint of the ZIP Code boundary) (U.S. only). |
| SX   | Matched to a street intersection.                                                 |
| S0   | Single close match, no coordinates available.                                     |
### Eight Additional Precision Characters

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>Matched to house number.</td>
</tr>
<tr>
<td>P</td>
<td>Matched to street prefix.</td>
</tr>
<tr>
<td>N</td>
<td>Matched to street name.</td>
</tr>
<tr>
<td>T</td>
<td>Matched to street type.</td>
</tr>
<tr>
<td>S</td>
<td>Matched to street suffix.</td>
</tr>
<tr>
<td>C</td>
<td>Matched to city name.</td>
</tr>
<tr>
<td>Z</td>
<td>Matched to postal code.</td>
</tr>
<tr>
<td>A or U</td>
<td>Match came from MapMarker Address Dictionary (A) or customized user dictionary (U).</td>
</tr>
</tbody>
</table>

### Z Category: Postal Code Match

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z3</td>
<td>ZIP+4 centroid match.</td>
</tr>
<tr>
<td>Z2</td>
<td>ZIP+2 centroid match.</td>
</tr>
<tr>
<td>Z1</td>
<td>Postal code centroid match.</td>
</tr>
<tr>
<td>Z0</td>
<td>Postal code match, no coordinates available.</td>
</tr>
</tbody>
</table>

### N Category: Non-matches

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>No close match.</td>
</tr>
<tr>
<td>NX</td>
<td>No close street intersection match.</td>
</tr>
<tr>
<td>ND</td>
<td>Data for given postal code or city/state is unavailable.</td>
</tr>
</tbody>
</table>
Troubleshooting

This chapter gives you a place to start if you encounter problems when you geocode with MapInfo Geocoding Extender for SQL Server.

In this chapter:

- Product Setup. .............................................. 114
- Error Codes. .................................................... 115
When trying to determine why you cannot geocode your record or table, begin by checking the system from the base product up. Be sure that you followed the installation order for the products.

**MapMarker**

Consider the following:

- Is MapMarker or MapMarker Plus 11.x or later installed?
- Can you geocode to it directly? Do you have data loaded? Use the sample table US_ADDR.tab in the \MapMarker\sampdata directory as a test.

**Geocoding Server**

Consider the following:

- Check the server connection.
- Verify that you can geocode an address.
- Does it geocode? If not, check that the settings and preferences match the working geocoding server. For example, if you are getting different results with the Geocoding Extender than you are with MapMarker Desktop, double-check that the Geocoding Extender preference settings match the settings in MapMarker Desktop. They may be different.
- Check that the server is actually running. Use the sample applications provided with MapMarker and Envinsa to determine if the server is running.

**MapInfo Geocoding Extender for SQL Server**

Consider the following:

- Does your account have privileges for the requested operation?
- Are your tables in the correct form?
- Have they been described in the country table?
- Make sure that all the columns have been described. Check spelling, syntax, and punctuation.
- Have you added the correct columns?
- Have you checked the default settings?
- Ensure that you have set up your data and settings correctly for the particular geocoding operation you wish to perform.
- Do you have a place to output your results?
- Have you fully qualified your tables?
- Are you using the correct database?
- .NET CLR must be enabled before running the Geocoding Extender installer.
## Error Codes

The following error codes may be returned by MapInfo Geocoding Extender for SQL Server. In the event of an error, verify that the geocoding server is running and test that it can access the Address Dictionary by geocoding an address. The error codes below can provide more information for MapInfo Technical Support if an error occurs.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No default preference.</td>
</tr>
<tr>
<td>2</td>
<td>Too many default preference entries.</td>
</tr>
<tr>
<td>3</td>
<td>Problem selecting default preference or named preference.</td>
</tr>
<tr>
<td>4</td>
<td>No preference named XXX found.</td>
</tr>
<tr>
<td>5</td>
<td>GeoAddress not geocoded.</td>
</tr>
<tr>
<td>6</td>
<td>No Candidate at index.</td>
</tr>
<tr>
<td>7</td>
<td>No AddressRange at index.</td>
</tr>
<tr>
<td>8</td>
<td>No input country was specified.</td>
</tr>
<tr>
<td>9</td>
<td>No server URL was specified.</td>
</tr>
<tr>
<td>10</td>
<td>Too many named preferences found.</td>
</tr>
<tr>
<td>11</td>
<td>Country table does not exist.</td>
</tr>
<tr>
<td>12</td>
<td>Cannot instantiate a UDT object.</td>
</tr>
<tr>
<td>13</td>
<td>No AddressRange unit at index.</td>
</tr>
<tr>
<td>14</td>
<td>Unable to assign the CandidateSet back to the GeoAddress.</td>
</tr>
<tr>
<td>15</td>
<td>Unable to assign the AddressRangeSet back to the GeoAddress.</td>
</tr>
<tr>
<td>16</td>
<td>Unable to assign the AddressRangeUnitSet to the GeoAddress.</td>
</tr>
<tr>
<td>17</td>
<td>The OutputTableInfo is invalid.</td>
</tr>
<tr>
<td>18</td>
<td>Invalid log file specified in MM_Preference.</td>
</tr>
<tr>
<td>19</td>
<td>Error accessing input records.</td>
</tr>
<tr>
<td>21</td>
<td>Unable to write to the output table.</td>
</tr>
<tr>
<td>22</td>
<td>Unable to update the GeoAddress table.</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>23</td>
<td>Unable to translate the MMPreference.</td>
</tr>
<tr>
<td>25</td>
<td>Unable to translate the Candidate.</td>
</tr>
<tr>
<td>26</td>
<td>Unable to translate the AddressRange.</td>
</tr>
<tr>
<td>27</td>
<td>Unable to translate the AddressRangeUnit.</td>
</tr>
<tr>
<td>28</td>
<td>SRID is invalid.</td>
</tr>
<tr>
<td>29</td>
<td>Invalid additional fields key in MMPreference.</td>
</tr>
<tr>
<td>30</td>
<td>Invalid additional fields key in address.</td>
</tr>
<tr>
<td>31</td>
<td>Additional field value not provided in MMPreference.</td>
</tr>
<tr>
<td>32</td>
<td>Additional field value not provided in address.</td>
</tr>
<tr>
<td>33</td>
<td>Unable to access GeoAddress data.</td>
</tr>
<tr>
<td>34</td>
<td>No Geocoder assigned to Extender.</td>
</tr>
<tr>
<td>36</td>
<td>No output table name in OutputTableInfo.</td>
</tr>
<tr>
<td>37</td>
<td>No output table key column in OutputTableInfo.</td>
</tr>
<tr>
<td>38</td>
<td>Unable to find entry for output table/geometry column in ALL_SDO_GEOM_METADATA.</td>
</tr>
<tr>
<td>39</td>
<td>Multiple entries in ALL_SDO_GEOM_METADATA for output table/geometry column.</td>
</tr>
<tr>
<td>40</td>
<td>No input table name in InputTableInfo.</td>
</tr>
<tr>
<td>41</td>
<td>No input table key column in InputTableInfo.</td>
</tr>
<tr>
<td>42</td>
<td>No input columns given in InputTableInfo.</td>
</tr>
<tr>
<td>43</td>
<td>Error accessing connection to database.</td>
</tr>
<tr>
<td>45</td>
<td>Unable to access GeoAddress table.</td>
</tr>
<tr>
<td>46</td>
<td>Record failed to be geocoded.</td>
</tr>
<tr>
<td>47</td>
<td>Failed to update the geocoded record.</td>
</tr>
<tr>
<td>51</td>
<td>A geocoder error occurred.</td>
</tr>
<tr>
<td>52</td>
<td>A geocoder server error occurred.</td>
</tr>
<tr>
<td>53</td>
<td>No GeoAddress table name given.</td>
</tr>
</tbody>
</table>
### GeoEngine Error Code and Description

<table>
<thead>
<tr>
<th>GeoEngine Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GeoEngine ERROR: MALLOC_ERR.</td>
</tr>
<tr>
<td>2</td>
<td>GeoEngine ERROR: MEMORY_CORRUPTED_ERR.</td>
</tr>
<tr>
<td>3</td>
<td>GeoEngine ERROR: BAD_PARAM_ERR.</td>
</tr>
<tr>
<td>4</td>
<td>GeoEngine ERROR: FILE_NOT_FOUND_ERR.</td>
</tr>
<tr>
<td>5</td>
<td>GeoEngine ERROR: FILE_OPEN_ERR.</td>
</tr>
<tr>
<td>6</td>
<td>GeoEngine ERROR: FILE_READ_ERR.</td>
</tr>
<tr>
<td>7</td>
<td>GeoEngine ERROR: FILE_WRITE_ERR.</td>
</tr>
<tr>
<td>8</td>
<td>GeoEngine ERROR: INDEX_OPEN_ERR.</td>
</tr>
<tr>
<td>9</td>
<td>GeoEngine ERROR: INDEX_ACCESS_ERR.</td>
</tr>
<tr>
<td>10</td>
<td>GeoEngine ERROR: BAD_DATABASE_ERR.</td>
</tr>
<tr>
<td>11</td>
<td>GeoEngine ERROR: END_OF_DATA.</td>
</tr>
<tr>
<td>12</td>
<td>GeoEngine ERROR: BAD_POSTAL_CODE.</td>
</tr>
<tr>
<td>13</td>
<td>GeoEngine ERROR: UNINIT_DB_COMPONENT.</td>
</tr>
<tr>
<td>14</td>
<td>GeoEngine ERROR: BAD_INPUT_ADDRESS.</td>
</tr>
<tr>
<td>15</td>
<td>GeoEngine ERROR: NO_DATAAVAILABLE.</td>
</tr>
<tr>
<td>16</td>
<td>GeoEngine ERROR: INVALID_ACCESS_ERR.</td>
</tr>
<tr>
<td>18</td>
<td>GeoEngine ERROR: BAD_LICFILE_ERR.</td>
</tr>
<tr>
<td>19</td>
<td>GeoEngine ERROR: EXCEEDED_LIMIT.</td>
</tr>
<tr>
<td>20</td>
<td>GeoEngine ERROR: USER_DICT_SCAN_DONE.</td>
</tr>
<tr>
<td>XXXX</td>
<td>Unknown error:</td>
</tr>
</tbody>
</table>
This appendix provides a glossary of geocoding, census, and postal terms.
Glossary of Terms

Candidate
Record from the MapMarker Address Dictionary that is considered a potential match for the input address, based on its score.

Carrier Route
USPS address element that describes a given mail delivery or collection route within a 5-digit ZIP Code. There are five possible Carrier Route codes: Bnnn = P.O. Box; Hnnn = Highway Contract; Rnnn = Rural Route; Cnnn = City Delivery; and Gnnn = General Delivery (where n = a number).

CASS Mode
A strict manner of geocoding records, based on the standards set by the USPS Coding and Accuracy Support System (CASS), which strives for accurate addresses for mail pieces. Geocoding in this way matches records to the exact house number, street name and ZIP Code. Other geocoding preferences are overridden.

Census Block
United States Census Bureau code that uniquely identifies a geographic area for census reporting. Up to a 15-digit alphanumeric code can be returned for a successful match, depending on the result code. The census value is interpreted as follows:

\[
\text{SSCCCTTTTTTGBB(B)}
\]

where

\[
\begin{align*}
S &= \text{State Federal Information Processing Standard (FIPS) code (2 characters)} \\
C &= \text{County FIPS code (3 characters)} \\
T &= \text{Census Tract (6 characters)} \\
G &= \text{Census Block Group (1 character)} \\
B &= \text{Census Block (typically 2 characters. The third character means the block was split from another block.)}
\end{align*}
\]

S5 matches return the full code to the Census Block level. S1, S2, S3, Z1, Z2 and Z3 matches return the Block Group code (12 digits). No value is returned for S4 matches.

Check Digit
A one-digit code that together with Delivery Point is used to form the delivery point bar code on United States mail pieces.

City Subdivision
A subdivision of the city. Used in the United States for Puerto Rican urbanization areas. It is also called Area4 or AreaName4 in the MapInfo Geocoding Extender for SQL Server.

City
Name of city. It is also called Area3 or AreaName3 in the MapInfo Geocoding Extender for SQL Server. In Envinsa, it is called ‘municipality’.

Country Division
Name of subdivision of a country. In the United States the country division is a state. It is also called Area1 or AreaName1 in the MapInfo Geocoding Extender for SQL Server.
**Delivery Point**
A two-digit code that along with the Check Digit is used to form the delivery point barcode on United States mail pieces.

**Firm**
Name of the firm, business, or organization at the address.

**Locatable Address Conversion System (LACS)**
A one-character code that indicates the record can be converted from a United States rural route address to a city-style address, using a separate USPS conversion product.

**MapMarker Address Dictionary**
The search dictionary used for matching United States addresses during geocoding in the MapMarker and MapMarker Plus U.S. products.

**Postal Code Add-on**
Additional postal zone identifier. In the United States it is a 4-digit extension to the 5-digit ZIP Code.

**Postal Code**
Unique identifier for postal mailing zones. In the United States it is a 5- or 9-digit ZIP Code.

**Precision**
Indicates the type of match candidate: Street level (20), street intersection (30), shape path (10, point on segment with no address ranges), or ZIP Code centroid (1, 2, or 3 for ZIP Code, ZIP+2 and ZIP+4 centroids, respectively). The numbers in parentheses refer to the precision's return value.

**Range**
House numbers along a street segment. Can be even, odd or both on each side of the street.

**Result Code**
A 10-character alphanumeric code that describes the type and accuracy of the geocoding match. See Result Codes Defined in Chapter 8 on page 109 for more information.

**Score**
The weighted rank given to a candidate, based on geocoding preferences set for the engine and the quality of the input address.

**Street**
Name of the street.
MapInfo SpatialWare for SQL Server allows the storage, access, management, and manipulation of spatial data as a standard component of a data set. You can write to the SpatialWare geometry formats in SQL Server.

MapInfo Geocoding Extender for SQL Server fully supports SpatialWare for SQL Server. SpatialWare support enables you return a geometry column as the location when you geocode. You can also retrieve spatial objects as the location of an address candidate.

This appendix explains what you should know to use SpatialWare tables with MapInfo Geocoding Extender for SQL Server

In this appendix:

- Integration with SpatialWare for SQL Server 2005 .................. 124
- Working with SpatialWare .................................................. 124
Integration with SpatialWare for SQL Server 2005

This section provides a brief overview of how MapInfo SpatialWare for SQL Server is integrated with MapInfo Geocoding Extender for SQL Server.

In SpatialWare for SQL Server, geometry can be represented as a user-defined data type (UDDT), called ST_SPATIAL. It is interpreted and managed by SpatialWare for SQL Server.

In MapInfo Geocoding Extender for SQL Server, SpatialWare geometry types and columns are handled in the following ways:

- When you geocode an address, you can choose to get the location as x and y columns, or as a geometry column.
- In the Candidate UDT, the GetSQLSpatialObject() method enables you to retrieve the spatial object as the location of the address candidate.
- When you geocode a table of addresses, the OutputTableInfo UDT can define the geometry column as the output of the address location.

Working with SpatialWare

If you have a MapInfo SpatialWare license and wish to use SpatialWare with MapInfo Geocoding Extender for SQL Server, please consider the following guidelines.

Before you start, please refer to the SpatialWare for SQL Server documentation on how to prepare your database and tables for SpatialWare. This will involve creating spatial tables, spatializing existing tables, populating them and creating R-tree indexes to enable spatial querying.

Before geocoding with the Geocoding Extender, do the following:

- You must ensure that your tables have x and y columns to insert the geocoded coordinates and also a column into which the SpatialWare sw_geometry can be inserted.
- If you have an existing SpatialWare table you wish to use with MapInfo Geocoding Extender for SQL Server, you must add a column into which the SpatialWare sw_geometry can be inserted.
- SpatialWare tables must have a primary key (e.g., sw_member) which you must set before performing geocoding procedures on your data.

Please see the MapInfo SpatialWare for SQL Server documentation for complete information on how to use SpatialWare.
MapInfo Geocoding Extender for SQL Server comes with several SQL scripts that demonstrate how to perform various geocoding tasks on your SQL Server tables.

This appendix describes the function of each script and explains how to run it.

In this appendix:

- Geocoding Demo Scripts ..............................126
To learn how to geocode using MapInfo Geocoding Extender for SQL Server, look at the demos in the GESS2.0\demo directory.

They include:

**findLocation.sql** – This script is a demo of how to use the FindLocation and the FindLocationWithPrefName stored procedures.

**geocodeAddress.sql** – This script is a demo of how to use the GeocodeAddress and the GeocodeAddressWithPrefName stored procedures.

**geocodeAddressLastLine.sql** – This script is a demo of how to use the GeocodeAddressLastLine and the GeocodeAddressLastLineWithPrefName stored procedures.

**singleFields.sql** – This script is a demo of how to use the Single_Fields and the Single_FieldsWithPrefName stored procedures.

**singleGeoAddr.sql** – This script is a demo of how to use the Single_GeoAddr and the Single_GeoAddrWithPrefName stored procedures.

**CENSUS_BLOCK.sql** – This script is a demo of how to use a country specific address field, such as census block, and write it to the output table.

**getSpatialObj.sql** – This script is a demo of how to get a spatial object that represents the address location from the geocoding.

**result.sql** – This spatial object is a binary that can be read and managed by MapInfo Spatialware.

**tableFields.sql** – This script is a demo of how to use the Table_Fields and Table_FieldsWithPrefName stored procedures.

**tableGeoAddr.sql** – This script is a demo of how to use the Table_GeoAddr and Table_GeoAddrWithPrefName stored procedures.
Preference Comparison

This appendix gives a comparison table of the preferences for MapInfo Geocoding Extender for SQL Server, MapMarker server, and Envinsa server.

In this appendix:

- preference comparison table. .......................... 128
The following table lists the preferences for MapInfo Geocoding Extender for SQL Server, and the corresponding preferences MapMarker server and Envinsa server.

For country-specific preferences, the Geocoding Extender uses the MM_Preference UDT object to carry them. The MM_Preference UDT object should be set as the preference parameter in each geocoding stored procedure.

For MapMarker servers, please refer to your MapMarker product for the exact preference key name. You must set the key and its value in the AddFields field of the MM_Preference UDT, and then pass it in as the preference parameter for each geocoding stored procedure. Country-specific preferences cannot be stored in the mmge_preferences table.

In Envinsa, the geocoding server uses LocalGeocodePreference to carry the country-specific preference like case mode. LocalGeocodePreference is equivalent to AddFields in the MM_Preference UDT.

The MapMarker or Envinsa server preferences are the default settings. They may be different in different countries depending on whether you are using a country version of MapMarker. The MM_Preference UDT content passed in the geocoding stored procedures will overwrite the preference values in MapMarker server or Envinsa server.

<table>
<thead>
<tr>
<th>MapInfo Geocoding Extender for SQL Server</th>
<th>MapMarker</th>
<th>Envinsa</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>COUNTRYTABLE</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>MUSTMATCHADDRESSNUMBER</td>
<td>MustMatchAddrNum</td>
<td>MustMatchAddrNum</td>
</tr>
<tr>
<td>MUSTMATCHMAINADDRESS</td>
<td>MustMatchMainAddr</td>
<td>MustMatchMainAddr</td>
</tr>
<tr>
<td>MUSTMATCHPOSTAL</td>
<td>MustMatchPostal</td>
<td>MustMatchPostalCode</td>
</tr>
<tr>
<td>MUSTMATCHINPUT</td>
<td>MustMatchInput</td>
<td>MustMatchInput</td>
</tr>
<tr>
<td>MUSTMATCHAREANAME1</td>
<td>MustMatchAreaName1</td>
<td>MustMatchCountrySubdivision</td>
</tr>
<tr>
<td>MUSTMATCHAREANAME2</td>
<td>MustMatchAreaName2</td>
<td>MustMatchCountrySecondarySubdivision</td>
</tr>
<tr>
<td>MUSTMATCHAREANAME3</td>
<td>MustMatchAreaName3</td>
<td>MustMatchMunicipality</td>
</tr>
<tr>
<td>MUSTMATCHAREANAME4</td>
<td>MustMatchAreaName4</td>
<td>MustMatchMunicipalitySubdivision</td>
</tr>
<tr>
<td>MAXCANDIDATES</td>
<td>MaxCandidates</td>
<td>MaxCandidatesReturned</td>
</tr>
<tr>
<td>MAXRANGESPERCANDIDATE</td>
<td>MaxRanges</td>
<td>maxRanges</td>
</tr>
<tr>
<td>MAXRANGEUNITSPERRANGE</td>
<td>MaxRangeUnits</td>
<td>maxRangeUnits</td>
</tr>
<tr>
<td>POSTALFALLBACK</td>
<td>FallbackToPostal</td>
<td>fallbackToPostal</td>
</tr>
<tr>
<td>MapInfo Geocoding Extender for SQL Server</td>
<td>MapMarker</td>
<td>Envinsa</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>GEOGRAPHYFALBACK</td>
<td>FallBackToGeographic</td>
<td>fallbackToGeographic</td>
</tr>
<tr>
<td>CORNEROFFSETUNIT</td>
<td>CornerOffsetUnits</td>
<td>cornerOffsetUnits</td>
</tr>
<tr>
<td>CORNEROFFSETDISTANCE</td>
<td>CornerOffset</td>
<td>OffsetFromCorner</td>
</tr>
<tr>
<td>STREETOFFSETUNIT</td>
<td>StreetOffsetUnits</td>
<td>StreetOffsetUnits</td>
</tr>
<tr>
<td>STREETOFFSETDISTANCE</td>
<td>StreetOffset</td>
<td>OffsetFromStreet</td>
</tr>
<tr>
<td>REQUESTSRID</td>
<td>ClientCoordinateSystem</td>
<td>N/A</td>
</tr>
<tr>
<td>CLOSEONLY</td>
<td>CloseMatchesOnly</td>
<td>closeMatchesOnly</td>
</tr>
<tr>
<td>BATCHSIZE</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DEFAULTPREFERENCE</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>LOGFILE</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>LOGLEVEL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Index

A
adding a country 101–102
address dictionary (definition) 121
address matching 10
AddressRange object 36–37
AddressRangeUnit object 37
administrator rights 20
assigning default preferences 94

C
candidate (definition) 120
Candidate object 37–39
carrier route (definition) 120
CASS mode (definition) 120
Census Block (definition) 120
check digit (definition) 120
city subdivision (definition) 120
CommonAddress object 39–41
CommonTable object 41–43
country division (definition) 120
country tables 99–100, 102–103, 105–107
creating preferences 91–93

D
database triggers 14
default preference 94–95
deleting a country 102
deleting preferences 93–94
Delivery point (definition) 121
demo scripts 126
documentation 16

E
error codes 115–117
examples
FindLocation stored procedure 68
GeocodeAddress stored procedure 77–78
GeocodeAddressLastLine stored procedure 81–82
Single_Fields stored procedure 72
Single_GeoAddr stored procedure 74–75
Table_Fields stored procedure 58–60
Table_GeoAddr stored procedure 63
triggers 83–85

F
file locations 28
FindLocation stored procedure 66–68

G
GeoAddress object 43–45
GeocodeAddress stored procedure 75–78
GeocodeAddressLastLine stored procedure 78–82
géocoding
defining requirements 32
returning best candidate 66
returning multiple candidates 72–73
single addresses 33
tables of addresses 33
géocoding demo scripts 126
géocoding servers 14, 20–21, 114
géocoding single addresses
FindLocation stored procedure 66–68
GeocodeAddress stored procedure 75–78
GeocodeAddressLastLine stored procedure 78–82
Single_Fields stored procedure 69, 71–72
Single_GeoAddr stored procedure 73–75
triggers 82–85
getting country information 103–104
getting country server information 107
getting current preference settings 98–99
getting default preference 95–96
glossary of terms 120–121

I
InputTableInfo object 45–46
installation
administrator rights 20
errors 28
prerequisite components 21
Index

L

LACS (definition) 121

M

managing country tables 100, 102–103, 105–107
MapInfo SpatialWare 21
matching addresses
  with geographic data 10
Microsoft SQL Server 15
MM_Preference object 46–50
mmge_preferences table 88–91
modifying preferences 96–97
multi-threading 14

N

N category result codes 112
new features 16

O

operating systems 20
OutputTableInfo object 50–51

P

precision (definition) 121
precision characters in result codes 112
preferences
  comparison table 128
  managing settings 91–99
  mmge_preferences table 88–91
  procedures 13
pre-installation 21

R

range (definition) 121
reinstalling 29
requirements
  system 20–21
result codes 110, 112, 121
  address point interpolation 111

S

S category result codes 111
samples 15
score (definition) 121
setting country information 105, 107
setting default preference 94–95
setting up tables 54
single addresses
geocoding 33
Single_Fields stored procedure 69, 71–72
Single_GeoAddr stored procedure 73–75
SpatialWare support 15, 21, 123–124
SQL Server 15
system requirements 20–21

T

Table_Fields stored procedure 55
Table_GeoAddr stored procedure 60–63
tables
  country 99–100, 102–103, 105–107
  geocoding 33
  mmge_preferences 88–91
  preparation 32
  setup 54
  Table_Fields procedure 55–60
Table_GeoAddr stored procedure 60–63
threading 14
triggers 14, 82–85

U

uninstallation 29
user rights in SQL Server 22

Z

Z category result codes 112