MapMarker 3.x Application Programming Interfaces

Overview

This document contains the function calls and sample code for developers to create client/server or standalone geocoding applications.

Important: The information presented here is current as of the latest shipping version of MapMarker 3.4. It does **not** include any information related to MapMarker Beta release.

This document is organized as follows:

Custom Applications and CD-ROM Unlocking

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

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

In the first situation, you, as developer, unlock the data by installing MapMarker and calling MapInfo for an access code. At this time one or more license files are created in \mapinfo\mapmarkr\data. To deploy your custom application, you must install the MapMarker engine (mm32v3.dll), the Address Dictionary, the license files, and your application on your client's system. The client's and your serial number for MapMarker must be the same since we embed your serial number into the license files at the time of unlocking. The serial number is stored in the Windows Registry at HKEY_LOCAL_MACHINE \SOFTWARE\MAPINFO\ MAPMARKR\ USER\SerialNumber. It is also stored in HKEY_CURRENT_USER.

In the second scenario, the client unlocks the data by obtaining an access code from MapInfo during the installation of the standard MapMarker product. At that time the license file(s) is installed on the client's system. This give the client access to MapMarker and the Address Dictionary. You then install your application on the client's system. No verification of serial numbers is required since the data is already unlocked and in place.

Chapter 1: MapMarker GeoEngine API

This chapter provides information for developers who wish to create customized geocoding applications based on the MapMarker Geocoding Engine Application Programming Interface.

This is the C API for the MapMarker geocoding engine used for adding batch geocoding capabilities to your application. Included are the function calls, programming notes and sample code to assist you. The MapMarker geocoding engine and its API are packaged as a 32-bit Dynamic Link Library (mm32v3.dll).

In This Chapter . . .

Creating a Geocoding Application

Overview

The MapMarker Geocoding Engine API provides all the functionality needed to create a full scale geocoding application. The API provides calls to do batch and interactive geocoding, as well as calls to do browsing and intersection matching. The easiest way to begin writing geocoding applications is to customize the included sample applications.

Note: The API does not support the creation of MapInfo tables or the creation of MapInfo point objects. However, this functionality is available to developers of MapBasic applications using standard MapBasic functions.

The MapMarker Geocoding Engine API supports the following:

- Geocoding in batch mode or interactively
- Matching to street–level, ZIP centroid, place name and street intersections
- Matching to the MapMarker Address Dictionary and customized dictionaries
- Creating customized user dictionaries
- Converting to and from NAD83 and NAD27
- Returning candidate information found in the MapMarker Address Dictionary
- Setting matching restrictions and system preferences
- Geocoding with an evaluation copy of MapMarker

The following sections will detail how to write an "ANSI C" geocoding application. The API is available on Windows, HP–UX, and Solaris 2.4. The 32-bit Windows DLL was built using Microsoft's Visual C++ V 4.1. This DLL should be compatible with most Windows compilers.

Concepts

All of the functions in the API return a short value. This short value can be used to determine whether the call was successful or not. If the call was successful, then the value SUCCESS is returned. If not, then one of the error codes in geoerror.h will be returned. The caller should be sure to check the return value from a function every time. If one step of the geocoding process fails, then the remaining calls may or may not be valid, and could lead to confusing results.

The API uses the term postalCode and postalAddOnCode instead of ZIP Code and ZIP+4. This is for possible future internationalization.

Within the sample source code [. . .] will appear. This signifies that some source code was left out because it did not pertain to the example. To see the complete set of source code consult the "Programming Use Example" at the end of this chapter.

Initialization

Before any calls in the API can be used (GeoEngDbCheckAvailability is the only exception) the Geocoding Engine needs to be initialized. Initialization involves finding the Address Dictionary files, as well as reading in the parsing and matching support files (geo_usa.*). The call GeoEngInit is used to initialize the engine. It can be used to initialize any combination of street, ZIP Code, or user dictionary level geocoding. GeoEngInit takes as a parameter a pointer to a GEO_ENG_HANDLE. This handle is used by every call in the API with the exception of GeoEngDbCheckAvailability. It is how the Geocoding Engine keeps track of the current state of the application's geocoding progress.

Once the application has completed geocoding, the GeoEngKill is used to terminate geocoding. It simply closes all files and frees any resources. It is important to note that GeoEngKill needs to be called before another call to GeoEngInit.

The call GeoEngDbCheckAvailability can be used before the call to GeoEngInit to determine the availability of the various geocoding dictionaries. The call will also try to initialize the parsing and matching components of the engine so that if any of the support files are not available an error code will be returned.

The following sample code obtains the Address Dictionary database path from the user's mapinfo.ini file, and then tries to initialize street and centroid geocoding. For other platforms the code simply asks the user to enter the database path at the command line prompt.

```
/* For WINDOWS include these */
#if defined(_WINDOWS)||defined(_WIN32)
  #include <windows.h>
#endif

/* Include the files needed for the Geocoding Engine. */
#ifdef _WINDOWS
  #define GEODLL
#endif

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

/* Prototypes for local functions */
short InitializeEngine(pGEO_ENG_HANDLE);
[ . . . ]
```

```
/*
 * InitializeEngine - Fires up the MapMarker Geocoding Engine. For
Windows it
 *   consults the user ini file. For other platforms it asks for
input.
 */
short InitializeEngine(pGEO_ENG_HANDLE pGeoEngHandle)
{
    char dbPath[256]     = "";
    char *dbName      = "geo_usa";
    GEO_ENG_CFG_PARMS geoEngParms;
    short retCode;
    short dbFlags;

    printf("Initializing the Geocoding Engine. This could take
awhile.\n");
   printf("Please enter the full path of the MapMarker address
dictionary\n");
   gets (dbPath);

    /*
     * Check availability, we only initialize if STREET_DB and
CENTROID_DB
     *    are available.
     */
    retCode = GeoEngCheckDbAvailability (dbPath, dbName, NULL,
&dbFlags);
    if (retCode != SUCCESS)
    {
        fprintf (stderr, "Error %d checking MapMarker dictionary
                                  availability\n",retCode);
        return FAILURE;
    }
    if (!(dbFlags & STREET_DB))
    {
        fprintf(stderr, "Geocoder Database Street files not
available\n");
        fprintf(stderr, "Please try again with the correct dbPath\n");
        return FAILURE;
    }
    if (!(dbFlags & CENTROID_DB))
    {
        fprintf(stderr, "Geocoder Database Centroid files not
available\n");
        fprintf(stderr, "Please try again with the correct dbPath\n");
    return FAILURE;
    }
    /*
     * If I got here, then try and initialize the engine.
     */
    retCode = GeoEngInit(dbPath, dbName, NULL,
```

```
                    STREET_DB | CENTROID_DB, pGeoEngHandle);
    if (retCode != SUCCESS)
    {
        fprintf (stderr, "Error %d initializing MapMarker\n",retCode);
        return FAILURE;
    }
[ . . . ]
    return SUCCESS;
} /* end InitializeEngine function */
```

A note about include files. At the top of the sample application, the file geo.h, and geoerror.h are included. These source files are supplied with the Geocoding Engine source code. The geo.h and geostd.h files are needed to allow your application to reference all the defined constants, structures and prototypes for the API. The geoerror.h contains some specific constants for error codes returned by the API. The list in geoerror.h is not complete, but should cover most return codes that you would see in a typical application.

Setting Runtime Configuration Parameters

The API enables the application to control the behavior of the geocoding as well as the placement of geocoded points in relation to the matched segment. This behavior is controlled through the use of the GEO_ENG_CFG_PARMS structure.

```
typedef struct
{
  double          linearInset;
  double          perpendicularSetback;
  DIST_UNIT       distUnit;
  BOOLEAN         relaxHouseNumber;
  BOOLEAN         relaxPostalCode;
  BOOLEAN         relaxName;
  BOOLEAN         relaxFinance;
  BOOLEAN         relaxFinanceInState;
  BOOLEAN         matchIntersections;
  BOOLEAN         useCassMode;
  short           relaxDistance;
} GEO_ENG_CFG_PARMS, *pGEO_ENG_CFG_PARMS;
```

The relaxHouseNumber, relaxPostalCode, and relaxName members control the behavior of the geocoding operation.

If relaxPostalCode is TRUE, then the API will search the finance number of the given ZIP Code, as well as the finance number of the city and state. The matched address and the input address need not have the same ZIP Code. If the parameter is FALSE, then only the finance number of the ZIP Code is searched, and the match address and the input address need to have the same ZIP Code.

If relaxName is TRUE, then the API will do a SOUNDEX search for a given street name and will choose what it thinks is the best match. If the value is TRUE, then only an exact street name match will be considered acceptable.

The relaxHouseNumber member is used to tell the API that the matched address and the input address need not have the same house number. The API will place the matched coordinate at the closest (sequentially) house number on the matched street. If no house numbers are available for the street, then the API will geocode the input address to the midpoint of the shape path.

If relaxFinance is TRUE, then the API will search a wider area for the address. The size of this area is defined by relaxDistance. All finance areas that have centroids within relaxDistance will be searched as well. If relaxFinanceInState is TRUE, then the widened search area (indicated by relaxFinance being TRUE) will be limited to the current state. Finance areas in other states will not be searched.

If matchIntersections is TRUE, then the API will attempt to match the input address to street intersections.

If useCassMode is TRUE, then all of the other configuration parameters will be assigned their default CASS values. The default CASS values are:

linearInset = 25.0
perpendicularSetback = 20.0
distUnit = DIST_UNIT_FOOT
relaxHouseNumber = False
relaxPostalCode = True
relaxName = True
relaxFinance = False
relaxFinanceInState = False
matchIntersections = False
relaxDistance = 0

The distUnit, linearInset, and perpendicularSetback values are used to control the location of a point when geocoding requires interpolation along a street segment. The linear inset is a value that can be used to set all points a given distance from an intersection. This can be used to prevent the first house on cross streets from sitting on top of one another. The perpendicularSetback is used to move the interpolated point away from the center of the street. This keeps points on the left and right sides of the street segments.

There are two calls supplied for manipulating the configuration parameters. They are GeoEngSetCfgParms and GeoEngGetCfgParms. It is recommended that the application get the current values before changing the values. This way the application need only modify the values in the structure that it is going to change.

The following code sample gets the current values, and then relaxes names and ZIP Codes, and requires house numbers.

```
short InitializeEngine(pGEO_ENG_HANDLE pGeoEngHandle)
{
    char dbPath[256]     = "";
    char *dbName     = "geo_usa";
    GEO_ENG_CFG_PARMS geoEngParms;
    short retCode;
    short dbFlags;
[ . . . ]
        /*
         * Get the current configuration values and set them to what
we want.
         */
        retCode = GeoEngGetCfgParms (*pGeoEngHandle, &geoEngParms);
        if (retCode != SUCCESS)
        {
       fprintf (stderr, "Error %d reading configuration
parameters\n",retCode);
            return FAILURE;
        }
    else
        {
            geoEngParms.relaxHouseNumber = FALSE;
            geoEngParms.relaxName = TRUE;
            geoEngParms.relaxPostalCode = TRUE;
            retCode = GeoEngSetCfgParms(*pGeoEngHandle, &geoEngParms);
            if (retCode != SUCCESS)
            {
                fprintf (stderr, "Warning: Error %d setting
configuration
parameters\n",retCode);
                fprintf (stderr, "\tDEFAULT VALUES WILL BE USED\n");
            }
        }
    }
[ . . . ]
    return SUCCESS;
} /* end InitializeEngine function */
```

Matching the Address

The API function GeoEngMatchAddress is the routine that attempts to geocode the input address. The input address needs to be supplied to the application and must contain the street name and the ZIP Code (although if ZIP Codes are relaxed, city and state suffice). It returns SUCCESS if it was able to parse the address and tries to match candidates against the input address. The matchStatus parameter can be used to determine the extent to which the address was matched. If the matchStatus is SINGLE_MATCH, then a single good-street match that met all the relaxation criteria was found, and the application can then geocode the address using GeoEngGetCandidateCoords. If the matchStatus was MULTIPLE_MATCHES, then more than a single good-street match was found. At this point the application can choose to geocode the address or use some of the clerical functions described later to allow interactive geocoding of the address. A matchStatus of NO_MATCHES means that some candidates were found but none were close matches that met the relaxation criteria. At this point the application could choose to not geocode the address or once again use the clerical functions to allow interactive geocoding. The NO_CANDIDATES match result means there was nothing in the Address Dictionary even close to the input address. Browsing would be the only option at this point.

```
    short             matchStatus;
    GEO_ENG_HANDLE    geoEngHandle;
    ADDRESS           inputAddress;
    PARSED_ADDRESS    cleanAddress;
    short             retCode;

    retCode = GeoEngMatchAddress(geoEngHandle,&inputAddress,
                                 &cleanAddress,&matchStatus);
[ . . . ]
    else if (retCode == SUCCESS)
    {
        switch (matchStatus)
        {
            case SINGLE_MATCH:
                [ . . . ]
                break;
            case MULTIPLE_MATCHES:
                [ . . . ]
                break;
            case NO_MATCHES:
                [ . . . ]
                break;
            case NO_CANDIDATES:
                [ . . . ]
                break;
            case SINGLE_INTERSECT_MATCH:
```

```
                    [ . . . ]
                    break;
                case MULTIPLE_INTERSECT_MATCH:
                    [ . . . ]
                    break;
                case NO_INTERSECT_MATCHES:
                    [ . . . ]
                    break;
                case NO_INTERSECT_CANDIDATES:
                    [ . . . ]
                    break;
                    default:
                    printf("Invalid match status = %d\n",matchStatus);
                    break;
                }
        }
```

Getting Coordinates

The API provides two functions for getting coordinates. The routine GeoEngGetCandidateCoords can be used after a call to GeoEngMatchAddress that returns SUCCESS and a matchStatus of SINGLE_MATCH, or MULTIPLE_MATCHES. GeoEngGetCandidateCoords needs to be called before any new calls to GeoEngMatchAddress. This is because the API maintains a list of candidates for the most recent match attempt, and a call to GeoEngMatchAddress will replace the current candidate list.

```
    DOUBLE_POINT      location;
    short retCode;
    short precision;
    [ . . .]
    retCode = GeoEngGetCandidateCoords(geoEngHandle,0,
                                       &precision, &location);
    if (retCode != SUCCESS)
    {
        fprintf (stderr, "Error %d getting candidate
coordinates\n",retCode);
        return retCode;
    }
```

The other API call for getting coordinates can be used to obtain a ZIP Code or a ZIP+4 centroid. The call is GeoEngFindPostalCentroid. The call must be passed a valid ZIP Code, the Plus4 is optional. A NULL pointer or an empty string may be supplied for the ZIP+4.

```
    DOUBLE_POINT location;
    short retCode;
    short precision;
    retCode =
GeoEngFindPostalCentroid(geoEngHandle,pInputAddress->postalCode,
```

```
                                        pInputAddress->postalAddOnCode,
                                             &precision, &location, NULL);
    if (retCode != SUCCESS)
    {
        fprintf (stderr, "Error %d getting postal
centroid\n",retCode);
        return retCode;
    }
    else
    {
        printf ("MapMarker Point = %f longitude, %f latitude\n",
location.x, location.y);
    }
```

Getting the Candidates

The API provides the caller of GeoEngMatchAddress with the choice of viewing all the candidates that were found. This can be helpful if building an interactive geocoder. It can also be used to obtain candidate information for the given match such as Census Blocks, matched street address, matched ZIP+4, matched USPS last line information (city, state) as well as the score and how the various candidate fields matched.

The caller should first obtain the number of candidates in the current list. The call GeoEngGetCandidateCount is used here. The call returns the number of close matches, as well as the total number of candidates. A close match is a candidate that was close on all the match fields, and met the relaxation criteria (see Setting Runtime Configuration Parameters above). Once the total number of candidates is known, then the call GeoEngGetIndexedCandidate can be used to obtain the candidate address.

```
unsigned short    i;
unsigned short    closeMatchCount;
unsigned short    candidateCount;
CANDIDATE_ADDRESS matchAddress;
short             retCode;
short             precision;
DOUBLE_POINT      location;

[ . . .]
retCode = GeoEngGetCandidateCount(geoEngHandle,&closeMatchCount,
                                        &candidateCount);
if (retCode != SUCCESS)
{
   fprintf (stderr, "Error %d getting candidate count\n",retCode);
    return retCode;
}
[ . . .]
else
{
```

```
        for (i = 0; i < closeMatchCount; i++)
        {
            retCode = GeoEngGetIndexedCandidate(geoEngHandle, i,
&matchAddress);
            if (retCode != SUCCESS)
            {
                fprintf (stderr, "Error %d getting match
candidate\n",retCode);
                return retCode;
            }
        [ . . . ]
        }
    }
```

It is important to note that if GeoEngMatchAddress returns SUCCESS and a matchStatus of SINGLE_MATCH or MULTIPLE_MATCHES, then the GeoEngGetIndexedCandidate can always be used with an index 0, to obtain the best match.

The candidates returned all have a house number chosen to be the closest available house number on the street. If this is not good enough for the given application, or if the caller wishes to allow the user to choose the correct house number range, then the call GeoEngGetNextHouseRange can be used to examine all the house number ranges available. The house number ranges are a combination of TIGER street ranges, and USPS ZIP+4 ranges, so there could be a large number of them for a given candidate. The call GeoEngGetNextHouseRange can be called for each candidate and returns GEO_ENG_END_OF_DATA when done. The BOOLEAN parameter firstRange tells the API whether the caller wants the first range for this candidate, or the next. The following while loop demonstrates the use of this call.

```
    unsigned short        i;
     unsigned short     candidateCount;
     short              retCode;
     BOOLEAN            bFirst;
     HOUSE_RANGE        houseRange;

     for (i = 0; i < candidateCount; i++)
     {
     [ . . . ]
         /*
          * Now dump all the house ranges for the candidate
          */
         printf("\tHouse Ranges\n");
         bFirst = TRUE;
         while ((retCode = GeoEngGetNextHouseRange(geoEngHandle, i,
                                                   bFirst,
&houseRange)) == SUCCESS)
         {
```

```
                bFirst = FALSE;
                PrintHouseRange(&houseRange);
            }
            if (retCode != GEO_ENG_END_OF_DATA)
            {
                fprintf(stderr, "Error %d from GeoEngGetNextHouseRange\n",
                        retCode);
                return retCode;
            }
        [ . . .]
        }
    [ . . .]
    /*
     * PrintHouseRange - prints out the house range in a nice format.
     */
    void PrintHouseRange(pHOUSE_RANGE pHouseRange)
    {
        printf("\tFrom %10.10s To %10.10s ZIP Code %s-%s\n",
               pHouseRange->fromHouse, pHouseRange->toHouse,
               pHouseRange->postalCode, pHouseRange->postalAddOnCode);
    }
```

If the user decides that he or she wish to geocode the address using a different house range, then the call GeoEngGetIndexedCoords can be used. It takes the candidate index as well as the index of the house range, and returns the coordinate for the input house (closest house number on the range if available).

Browsing the Address Dictionary

The API allows an application to browse through the actual streets and places in the Address Dictionary. Since the dictionary is organized by finance number, the caller can only view streets and places in a given finance number at a time. The supplied ZIP Code is used to determine the finance number to browse. The calls to browse the dictionary are GenEngBrowseStreet, and GeoEngBrowsePlace. The caller needs to provide a street filter and a ZIP Code before calling these routines. This is done using the call GeoEngBrowseSetFilter. Any street or place that begins with the filter string, and is in the correct finance number for the given ZIP Code, will be returned. Thus if the filter was "MA" then all streets beginning with the letters "MA" in the finance number of the ZIP Code would be returned.

```
BOOLEAN         bFirst;
short           retCode;
char            browseFilter[256];
char            ZIPCode[256];
BROWSED_STREET browsedStreet;

[ . . .]
```

```
    bFirst = FALSE;
    retCode = GeoEngBrowseSetFilter(geoEngHandle, ZIPCode,
browseFilter);
    if (retCode != SUCCESS)
    {
        fprintf(stderr, "Error %d from GeoEngBrowseSetFilter\n",
retCode);
        return retCode;
    }
    while((retCode = GeoEngBrowseStreet(geoEngHandle, &browsedStreet))
== SUCCESS)
    {
        printf("\tStreet %s %s %s %s %s\n", browsedStreet.preDir,
               browsedStreet.preType, browsedStreet.name,
               browsedStreet.postType, browsedStreet.postDir);
    }
    if (retCode != GEO_ENG_END_OF_DATA)
    {
        fprintf(stderr, "Error %d from GeoEngBrowseStreet\n");
        return retCode;
    }
    [ . . .]
```

Include Files

The MapMarker Geocoding Engine API comes with include files for ANSI C applications. The files can be found in the "geoeng\include" subdirectory created when you install MapMarker.

For ANSI C, the application needs to include the files geo.h and geoerror.h. The geo.h include file contains all the function prototypes, defined constants and structure declarations needed when using the API. The file geoerror.h has error codes returned by the API calls. The list is not complete, but should cover all return codes seen in a typical application.

Compiling and Linking a Windows Application

A Window's ANSI C application needs to be compiled using the following command line flags:

/AL – Select the large memory model.

/ZP1 – Pack structure to 1 byte.

/D GEODLL – Define the constant GEODLL so prototype has the _export modifier.

The application then needs to link against the 32-bit MapMarker library (mm32v3.lib). It is placed in the "geoeng\lib" subdirectory that is created when MapMarker is installed. The application needs to increase the Stacksize to at least 20K. This can be done using the STACKSIZE argument in the Windows DEF file.

```
STACKSIZE       20480
```

MapMarker API Error Codes

For a complete list of error codes, see page 55.

Notes for MapBasic and Visual Basic Programmers

In addition to the C interfaces, we have provided interfaces to the MapMarker API for MapBasic and Visual Basic in the directory mapmarkr\geoeng\include. MapBasic programmers should include MIGEO.DEF in their programs, and Visual Basic programmers should use MIGEO.BAS. We have also provided MapBasic and Visual Basic example programs that demonstrate how to use the MapMarker API with these development environments. The examples are found in mapmarkr \geoeng\samples\. Programmers using other similar development environments should use these interface files and examples as a guide for developing interfaces in their environments.

There are a few special considerations when using the MapMarker API from MapBasic, Visual Basic or similar environments. You must take care that parameters passed to the MapMarker Engine (DLL) are declared and passed correctly. In MapBasic and Visual Basic, strings included in structures passed to DLLs must be declared with fixed length. However, strings passed as direct parameters to MapMarker routines should not be declared as fixed length. This means that a string passed to one routine as part of a structure and to another as

a direct parameter must be assigned to an intermediary local string variable of the correct type before being passed to the second routine.

MapBasic and Visual Basic programmers must also take care to always pass valid MapMarker engine pointers to the MapMarker engine. When you call GeoEngInit the returned value is a pointer to the memory used by that instance of the MapMarker engine. Though neither MapBasic nor Visual Basic have the concept of pointer variables, you can use a long integer (just an integer in MapBasic) to store the pointer so that it can be passed to the other routines that require it. The problem is that passing a variable that does not reference a valid instance of the MapMarker engine will cause an error or even a system crash. Things to watch out for include:

- Do not call any MapMarker routines after calling **GeoEngKill**.
- Be sure to pass the correct variable to the DLL.
- In Visual Basic, be sure to use the Option Explicit directive to force VB to prevent the use of undeclared variables. Without this setting, VB allows you to declare new variables simply by using them. In this situation, if you incorrectly type the name of the GeoEngine pointer in the parameter list of a call to a MapMarker routine, VB will create a new variable by that name with a value of zero and pass it to the DLL. This will usually result in a system crash. Note that Option Explicit is declared in the file MIGEO.BAS provided with MapMarker.

MapMarker Geocoding Engine Function Calls

GeoEngBrowsePlace() — Retrieve a place name from the Address Dictionary

Purpose

Retrieve the next place name from the Address Dictionary, which should previously have been positioned via **GeoEngBrowseSetFilter** without matching a user address. The current database position will be updated so that successive calls to this function will step through the place names that satisfy the filter string in the postal region that was specified in the most recent call to **GeoEngBrowseSetFilter**. Repeated use of this function will eventually fail when there are no more place names that satisfy the filter.

Syntax

short **GeoEngBrowsePlace** (GEO_ENG_HANDLE geoEngHandle,
pBROWSED_PLACE pPlaceName);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pPlaceName: A pointer to a variable that will receive the browsed place name. [Output]

Possible returns:.

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_END_OF_DATA	No more data available from the requested set.
GEO_ENG_NO_DATA_AVAILABLE	Internal database inconsistency.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures and Defines

```
/*
 * Defined constants for character array sizes in a BROWSED_PLACE structure
 */
#define MAX_PLACE_NAME_LEN     35
typedef struct {
  char name[MAX_PLACE_NAME_LEN];
} BROWSED_PLACE, *pBROWSED_PLACE;
```

GeoEngBrowseSetFilter() — Position within the Address Dictionary for browsing

Purpose

Set the filter for browsing within the Address Dictionary by means of the supplied filter string and postal region. This call must immediately precede the first use of **GeoEngBrowseStreet()** or **GeoEngBrowsePlace().** The next call to a function other than these will destroy the filter information. If this call is not issued, access to the database will be random. Calls to **GeoEngBrowseStreet()** cannot be interspersed with calls to **GeoEngBrowsePlace()**. If browsing of both is desired, you should complete browsing all streets or all places and then call **GeoEngBrowseSetFilter()** again before continuing with the other type of browsing.

Syntax

short **GeoEngBrowseSetFilter** (GEO_ENG_HANDLEgeoEngHandle,
char *pPostalCode,
char *pFilterString);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pPostalCode: A string containing the postal code. [Input]

pFilterString: A string whose contents specify the leading characters in all street or place names that can be retrieved using **GeoEngBrowseStreet** or **GeoEngBrowsePlace**. [Input]

NOTE: Once the filter has been used by calling **GeoEngBrowseStreet** or **GeoEngBrowsePlace** then it needs to be called again for any other browse calls.

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_DATABASE_ERR	Invalid database contents.
GEO_ENG_BAD_INPUT_ADDRESS	Input address is invalid or missing fields.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_BAD_POSTAL_CODE	Invalid postal code.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_FILE_READ_ERR	Error reading from database file.
GEO_ENG_INDEX_ACCESS_ERR	Error accessing database index.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

GeoEngBrowseStreet() — Retrieve a street address from the Address Dictionary

Purpose

Retrieve the next street address from the Address Dictionary. The database cursor should previously have been positioned via a call to **GeoEngBrowseSetFilter**. The current database position will be updated so that successive calls to this function will step through the streets that satisfy the filter string in the postal region that was specified in the most recent call to **GeoEngBrowseSetFilter.** Repeated use of this function will eventually fail when there are no more streets that satisfy the filter.

Syntax

short **GeoEngBrowseStreet** (GEO_ENG_HANDLEgeoEngHandle,
pBROWSED_STREET pStreetInfo);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pStreetInfo: A pointer to a variable that will receive the browsed street information from the database. The definition of this structure is given later in this document. [Output]

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_END_OF_DATA	No more data available from the requested set.
GEO_ENG_INDEX_ACCESS_ERR	Error accessing database index.
GEO_ENG_NO_DATA_AVAILABLE	Internal database inconsistency.
GEO_ENG_UNDEFINED_SEARCH	GeoEngBrowseSetFilter was not called.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures and Defines

```
/*
 *  Defined constants for the character array sizes in all the
BROWSED_STREET structure.
 */
#define MAX_STREET_PREDIR_LEN     3
#define MAX_STREET_PRETYPE_LEN    11
```

```
#define MAX_STREET_NAME_LEN       29
#define MAX_STREET_POSTTYPE_LEN   5
#define MAX_STREET_POSTDIR_LEN    3
typedef struct
{
  char preDir[MAX_STREET_PREDIR_LEN];
  char preType[MAX_STREET_PRETYPE_LEN];
  char name[MAX_STREET_NAME_LEN];
  char postType[MAX_STREET_POSTTYPE_LEN];
  char postDir[MAX_STREET_POSTDIR_LEN];
} BROWSED_STREET, *pBROWSED_STREET;
```

GeoEngCheckDbAvailability() — Check database availability

Purpose

Check which geocoding database components are available on a given search path using a specific dictionary name.

Syntax

short **GeoEngCheckDbAvailability** (char*pSearchPath,
char*pDbName,
char*pUserDictPath,
short*pDbFlags);

Parameters

pSearchPath: Directory search path to locate the dictionary files. This is a string containing directory paths separated by semicolons. [Input]

pDbName: Base name to use in constructing the names of the dictionary files. [Input]

pUserDictPath: full path to the user dictionary files. [Input]

pDbFlags: A pointer to an integer that will receive a value indicating which dictionary components were found. The value is the bitwise OR of the following defined constants and indicates which dictionary components can be initialized using the specified search path and Address Dictionary name. [Output]

STREET_DB
CENTROID_DB
USER_DB

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_MISSING_DATABASE_ERR	Address Dictionary file(s) not found.
GEO_ENG_ZIP_MASTER_FILE_NOT_FOUND_ERR	Zipmastr.cdb or Zipmastr.jdx not found.
GEO_ENG_DBQ_MISSING_LICFILE	License file not found.
GEO_ENG_INVALID_SERIAL_NUMBER	Invalid serial number.
GEO_ENG_PARSE_INIT_ERR	Parsing file (geo_usa.*) missing or bad.
GEO_ENG_MATCH_INIT_ERR	Matching file (geo_usa.*) missing or bad.
GEO_ENG_NADCON_MISSING_FILE	*.las/*.los files not found.

GeoEngFindPostalCentroid() — Get postal centroid coordinates

Purpose

Geocode by retrieving postal centroid coordinates, based on the supplied postal code and postal add on code.

Syntax

short **GeoEngFindPostalCentroid** (GEO_ENG_HANDLEgeoEngHandle,
char *pPostalCode,
char *pPostalAddOnCode,
short *pCoordPrecision,
pDOUBLE_POINT pLocation);
pCANDIDATE_ADDRESS pCandidate);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pPostalCode: A string containing the postal code. [Input]

pPostalAddOnCode: A string containing the postal add on code. If the postal add on code is not defined or not available, this parameter may be NULL. [Optional input]

pCoordPrecision: A pointer to a variable that will receive a value indicating the type of centroid that is returned. Higher numeric values indicate greater precision. The possible values follow. [Output]

1	ZIP code centroid
2	ZIP+2 centroid
3	ZIP+4 centroid
0	no match

pLocation: A pointer to a structure that will receive the geocoded coordinates. The definition of this structure is given later in this document. [Output]

pCandidate: A pointer to a the requested close address match. The definition of this structure is given later in this document. [Output]

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_DATABASE_ERR	Invalid database contents.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_BAD_POSTAL_CODE	Invalid postal code.

GEO_ENG_COORDS_NOT_AVAILABLE	Postal centroid coordinates not available.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_FILE_READ_ERR	Error reading from database file.
GEO_ENG_INDEX_ACCESS_ERR	Error accessing database index.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures and Defines

```
/*
 *  Precision levels for Coordinates returned by the engine.
 */
#define NO_CENTROID                     0
#define ZIPCODE_CENTROID                1
#define ZIPPLUS2_CENTROID               2
#define ZIPPLUS4_CENTROID               3
#define SHAPE_PATH_CENTER_COORDS        10
#define STREET_ADDRESS_COORDS           20
#define STREET_INTERSECT_COORDS         30
/*
 *  DOUBLE_POINT structure definition. It is how we return all
 *     coordinates.
 */
typedef struct {
  double x;
  double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
/*
 * Defined constants for the addressType field in the CANDIDATE_ADDRESS
structure.
 */
#define ADDRESS_TYPE_STREET           10
#define ADDRESS_TYPE_PLACE            11
#define ADDRESS_TYPE_ZIP              12
#define ADDRESS_TYPE_RURAL            13
#define ADDRESS_TYPE_HIGHWAY          14
#define ADDRESS_TYPE_POBOX            15
#define ADDRESS_TYPE_MILITARY         16
#define ADDRESS_TYPE_INTERSECTIO      17
/*
 * The following values are OR'd together to produce a fieldsMatchedValue
 *   value in the CANDIDATE_ADDRESS structure.
 */
#define NON_CLOSE_MATCH         1
#define CLOSE_MATCH             (1 << 1)
#define PREDIR_MATCH            (1 << 2)
#define NAME_EXACT_MATCH        (1 << 3)
#define POSTDIR_MATCH           (1 << 4)
#define TYPE_MATCH              (1 << 5)
#define POSTAL_CODE_MATCH       (1 << 6)
```

```
#define HOUSE_NUMBER_MATCH     (1 << 7)
#define USER_DICT_MATCH        (1 << 8)
/*
 *  Defined constants for the character array sizes in the ADDRESS
structure.
 */
#define MAX_BUSINESS_LEN               256
#define MAX_STREET_LEN                 256
#define MAX_CITY_LEN                    40
#define MAX_STATE_LEN                   40
#define MAX_POSTAL_CODE_LEN             10
#define MAX_POSTAL_ADDON_CODE_LEN       10
#define MAX_URBANIZATION_LEN            40
typedef struct
{
  char firm[MAX_BUSINESS_LEN];
  char street[MAX_STREET_LEN];
  char city[MAX_CITY_LEN];
  char state[MAX_STATE_LEN];
  char postalCode[MAX_POSTAL_CODE_LEN];
  char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
  char urbanization[MAX_URBANIZATION_LEN];
} ADDRESS, *pADDRESS;
/*
 *  Defined constants for the character array sizes in the CANDIDATE_ADDRESS
structure.
 */
#define MAX_CENSUS_BLOCK_LEN            20
#define MAX_STREET_LEN                 256
#define MAX_LAST_LINE_LEN               40
#define MAX_CARRIER_ROUTE_LEN            5
#define DELIVERY_POINT_LEN               3
#define CHECK_DIGIT_LEN                  2
#define MAX_LACS_LEN                     2

typedef struct
{
  short     addressType;
  short     fieldsMatchedValue;
  short     score;
  ADDRESS   candidateAddress;
  char      censusBlockId[MAX_CENSUS_BLOCK_LEN];
  char      tabbedAddress[MAX_STREET_LEN];
  char      primaryStreet[MAX_STREET_LEN];
  char      lastLine[MAX_LAST_LINE_LEN];
  char      deliveryPoint[DELIVERY_POINT_LEN];
  char      checkDigit[CHECK_DIGIT_LEN];
  char      lacs{MAX_LACS_LEN];
  char      carrierRoute[MAX_CARRIER_ROUTE_LEN];
} CANDIDATE_ADDRESS, *pCANDIDATE_ADDRESS;
```

GeoEngGetCandidateCoords() — Geocode a user address

Purpose

Geocode a user address using a specific match candidate and the best possible approximating house number for that candidate.

Geocode the user address that was previously supplied in the most recent call to **GeoEngMatchAddress**, using the best possible approximating house number that was identified for a specific match candidate. The house number to be used is the same one as reported in the house number field in the candidate when accessed via **GeoEngGetIndexedCandidate()**. This function may invalidate pointers previously returned by **GeoEngGetIndexedCandidate()** or **GeoEngGetNextHouseRange()**.

Syntax

short **GeoEngGetCandidateCoords** (GEO_ENG_HANDLE geoEngHandle,
unsigned short candidateIndex,
short *pCoordPrecision,
pDOUBLE_POINT pLocation);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

candidateIndex: The zero based index of the match candidate, relative to the number of available candidates, as reported by **GeoEngGetCandidateCount**. [Input]

pCoordPrecision: A pointer at a variable that will receive a value indicating the precision of the computed coordinates, with output values as follows. Higher numeric values indicate greater precision. [Output]

Values indicating precision of postal centroid
SHAPE_PATH_CENTER_COORDS
STREET_ADDRESS_COORDS

pLocation: A pointer to a structure that will receive the geocoded coordinates, which are computed using the current values of linearSetback and perpendicularInset. The definition of this structure is given later in this document. [Output]

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_CAND_INDEX	Match candidate index too large.
GEO_ENG_BAD_DATABASE_ERR	Invalid database contents.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_BAD_POSTAL_CODE	Invalid postal code.

GEO_ENG_COORDS_NOT_AVAILABLE	Postal centroid coordinates not available.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_FILE_READ_ERR	Error reading from database file.
GEO_ENG_INDEX_ACCESS_ERR	Error accessing database index.
GEO_ENG_INTERP_BAD_INSET	Invalid linear inset value.
GEO_ENG_INTERP_BAD_SETBACK	Invalid perpendicular setback value.
GEO_ENG_INTERP_HOUSE_ERR	Unable to interpolate house number.
GEO_ENG_MALLOC_ERR	Memory allocation failure.
GEO_ENG_NO_DATA_AVAILABLE	Internal database inconsistency.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures and Defines

```
/*
 *  Precision levels for Coordinates returned by the engine.
 */
#define NO_CENTROID                    0
#define ZIPCODE_CENTROID               1
#define ZIPPLUS2_CENTROID              2
#define ZIPPLUS4_CENTROID              3
#define SHAPE_PATH_CENTER_COORDS       10
#define STREET_ADDRESS_COORDS          20
#define STREET_INTERSECT_COORDS        30
/*
 *  DOUBLE_POINT structure definition. It is how we return all
 *     coordinates.
 */
typedef struct {
  double x;
  double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
```

GeoEngGetCandidateCount() — Get the number of match candidates

Purpose

Get the number of match candidates and/or the number of these that were highly similar to the input address, as evaluated by the most recent call to **GeoEngMatchAddress**. During matching, a weight is assigned to represent the result of matching against all of the component fields that make up the input and candidate addresses. The candidates are sorted in descending order of this assigned weight. This means that the candidates that are more similar to the input address will have the lower candidate index values. A threshold value for match weight has been heuristically defined. If compared addresses have an assigned weight equal to or above this value, they are considered to be highly similar.

Syntax

short **GeoEngGetCandidateCount** (GEO_ENG_HANDLE geoEngHandle,
unsigned short *pCloseMatchCount,
unsigned short *pCandidateCount);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pCloseMatchCount: A pointer at a variable that will receive the number of match candidates that closely matched the user input address. [Output]

pCandidateCount: A pointer at a variable that will receive the number of match candidates that were evaluated by the most recent call to **GeoEngMatchAddress** and that are available for subsequent review. [Output]

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.

GeoEngGetCfgParms() — Get current run time parameters

Purpose

Get current run time parameters for the geocoding engine.

Syntax

short **GeoEngGetCfgParms** (GEO_ENG_HANDLE geoEngHandle,
pGEO_ENG_CFG_PARMS pGeoEngParms);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pGeoEngParms: A pointer at a structure that will receive the current values of the parameters that affect the behavior of the geocoding engine. The definition of this structure is given later in this document. [Output]

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.

Structures

```
*
 * BOOLEAN with TRUE and FALSE.
 */
#define BOOLEAN short
/*
 * DIST_UNIT declaration and all the values it can have.
 */
typedef short DIST_UNIT, *pDIST_UNIT;
#define DIST_UNIT_FOOT            0
#define DIST_UNIT_DEGREE          1
#define DIST_UNIT_INCH            2
#define DIST_UNIT_LINK            3
#define DIST_UNIT_SURVEY_FOOT     4
#define DIST_UNIT_YARD            5
#define DIST_UNIT_ROD             6
#define DIST_UNIT_CHAIN           7
#define DIST_UNIT_MILE            8
#define DIST_UNIT_NAUTICAL_MILE   9
#define DIST_UNIT_MILLIMETER     10
#define DIST_UNIT_CENTIMETER     11
#define DIST_UNIT_METER          12
#define DIST_UNIT_KILOMETER      13
```

```
typedef struct
{
  double                linearInset;
  double                perpendicularSetback;
  DIST_UNIT             distUnit;
  BOOLEAN               relaxHouseNumber;
  BOOLEAN               relaxPostalCode;
  BOOLEAN               relaxName;
  BOOLEAN               relaxFinance;
  BOOLEAN               relaxFinanceInState;
  BOOLEAN               matchIntersections;
  BOOLEAN               useCassMode;
  short                 relaxDistance;
} GEO_ENG_CFG_PARMS, *pGEO_ENG_CFG_PARMS;
```

GeoEngGetIndexedCandidate() — Get a specified match candidate

Purpose

Get the specified match candidate that was evaluated by the most recent call to **GeoEngMatchAddress**. The match candidates are ordered by decreasing similarity to the original user address. This allows access to candidates, regardless of their similarity to the user address. Each match candidate will contain the best possible approximating house number for the user address in its house number field, considering all street house number ranges for that candidate. The user should not free the returned candidate pointer. This function may invalidate pointers previously returned by it or by **GeoEngGetNextHouseRange().**

Syntax

short **GeoEngGetIndexedCandidate** (GEO_ENG_HANDLE geoEngHandle,
unsigned short candidateIndex,
pCANDIDATE_ADDRESS pCandidate);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

candidateIndex: The zero based index of the match candidate desired, relative to the number of available candidates, as reported by **GeoEngGetCandidateCount**. [Input]

pCandidate: A pointer to a the requested close address match. The definition of this structure is given later in this document. [Output]

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_DATABASE_ERR	Invalid database contents.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_NO_DATA_AVAILABLE	Internal database inconsistency.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.
GEO_ENG_BAD_CAND_INDEX	Bad index number.

Structures and Defines

```
/*
 * Defined constants for the addressType field in the CANDIDATE_ADDRESS
structure.
 */
#define ADDRESS_TYPE_STREET           10
#define ADDRESS_TYPE_PLACE            11
#define ADDRESS_TYPE_ZIP              12
#define ADDRESS_TYPE_RURAL            13
#define ADDRESS_TYPE_HIGHWAY          14
#define ADDRESS_TYPE_POBOX            15
#define ADDRESS_TYPE_MILITARY         16
#define ADDRESS_TYPE_INTERSECTION     17
/*
 * The following values are OR'd together to produce a fieldsMatchedValue
 *   value in the CANDIDATE_ADDRESS structure.
 */
#define NON_CLOSE_MATCH         1
#define CLOSE_MATCH             (1 << 1)
#define PREDIR_MATCH            (1 << 2)
#define NAME_EXACT_MATCH        (1 << 3)
#define POSTDIR_MATCH           (1 << 4)
#define TYPE_MATCH              (1 << 5)
#define POSTAL_CODE_MATCH       (1 << 6)
#define HOUSE_NUMBER_MATCH      (1 << 7)
#define USER_DICT_MATCH         (1 << 8)
/*
 *  Defined constants for the character array sizes in the ADDRESS
structure.
 */
#define MAX_BUSINESS_LEN             256
#define MAX_STREET_LEN               256
#define MAX_CITY_LEN                  40
#define MAX_STATE_LEN                 40
#define MAX_POSTAL_CODE_LEN           10
#define MAX_POSTAL_ADDON_CODE_LEN     10
#define MAX_URBANIZATION_LEN          40
typedef struct
{
  char firm[MAX_BUSINESS_LEN];
  char street[MAX_STREET_LEN];
  char city[MAX_CITY_LEN];
  char state[MAX_STATE_LEN];
  char postalCode[MAX_POSTAL_CODE_LEN];
  char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
  char urbanization[MAX_URBANIZATION_LEN];
} ADDRESS, *pADDRESS;
/*
 *  Defined constants for the character array sizes in the CANDIDATE_ADDRESS
structure.
 */
```

```
#define MAX_CENSUS_BLOCK_LEN            20
#define MAX_STREET_LEN                 256
#define MAX_LAST_LINE_LEN               40
#define MAX_CARRIER_ROUTE_LEN            5
#define DELIVERY_POINT_LEN               3
#define CHECK_DIGIT_LEN                  2
#define MAX_LACS_LEN                     2

typedef struct
{
  short     addressType;
  short     fieldsMatchedValue;
  short     score;
  ADDRESS   candidateAddress;
  char      censusBlockId[MAX_CENSUS_BLOCK_LEN];
  char      tabbedAddress[MAX_STREET_LEN];
  char      primaryStreet[MAX_STREET_LEN];
  char      lastLine[MAX_LAST_LINE_LEN];
  char      deliveryPoint[DELIVERY_POINT_LEN];
  char      checkDigit[CHECK_DIGIT_LEN];
  char      lacs{MAX_LACS_LEN];
  char      carrierRoute[MAX_CARRIER_ROUTE_LEN];
} CANDIDATE_ADDRESS, *pCANDIDATE_ADDRESS;
```

GeoEngGetIndexedCoords() — Geocode a user address

Purpose

Geocode the user address that was previously supplied in the most recent call to **GeoEngMatchAddress(),** using a specific street house number range for a selected match candidate.

This function may invalidate pointers previously returned by **GeoEngGetIndexedCandidate()** or **GeoEngGetNextHouseRange()**.

Syntax

short **GeoEngGetIndexedCoords** (GEO_ENG_HANDLE geoEngHandle,
unsigned short candidateIndex,
unsigned short rangeIndex,
short*pCoordPrecision,
pDOUBLE_POINT pLocation);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

candidateIndex: The zero based index of the match candidate, relative to the number of available candidates, as reported by **GeoEngGetCandidateCount**. [Input]

rangeIndex:. The zero based index of the street house number range, relative to the full set of street house number ranges for the specified match candidate. [Input]

pCoordPrecision: A pointer at a variable that will receive a value indicating the precision of the computed coordinates, with output values as follows. Higher numeric values indicate greater precision. [Output]

Values indicating precision of postal centroid
SHAPE_PATH_CENTER_COORDS
STREET_ADDRESS_COORDS

pLocation: A pointer to a structure that will receive the geocoded coordinates, which are computed using the current values of linearSetback and perpendicularInset. The definition of this structure is given later in this document. [Output]

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_DATABASE_ERR	Invalid database contents.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_BAD_POSTAL_CODE	Invalid postal code.

GEO_ENG_BAD_RANGE_INDEX	Street house range index too large.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_INTERP_BAD_INSET	Invalid linear inset value.
GEO_ENG_INTERP_BAD_SETBACK	Invalid perpendicular setback value.
GEO_ENG_INTERP_HOUSE_ERR	Unable to interpolate house number.
GEO_ENG_MALLOC_ERR	Memory allocation failure.
GEO_ENG_NO_DATA_AVAILABLE	Internal database inconsistency.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures and Defines

```
/*
 *  Precision levels for Coordinates returned by the engine.
 */
#define NO_CENTROID                     0
#define ZIPCODE_CENTROID                1
#define ZIPPLUS2_CENTROID               2
#define ZIPPLUS4_CENTROID               3
#define SHAPE_PATH_CENTER_COORDS        10
#define STREET_ADDRESS_COORDS           20
#define STREET_INTERSECT_COORDS         30
/*
 *  DOUBLE_POINT structure definition. It is how we return all
 *     coordinates.
 */
typedef struct {
  double x;
  double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
```

GeoEngGetNextHouseRange () — Get the next street house range for a match candidate

Purpose

Get the next street house range for the specified match candidate from the candidates that were evaluated by the most recent call to **GeoEngMatchAddress**. The user should not free the returned house range pointer. This function will overwrite the structure referenced by any previous calls to it. If the first call to this function for a given match candidate does not specify firstRange equal to TRUE, results are unpredictable, and an error may be returned.

Syntax

short **GeoEngGetNextHouseRange** (GEO_ENG_HANDLE geoEngHandle,
unsigned short candidateIndex,
BOOLEAN firstRange,
pHOUSE_RANGE pHouseRange);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

candidateIndex: The zero based index of the match candidate, relative to the number of available candidates, as reported by **GeoEngGetCandidateCount**. [Input]

firstRange: TRUE if the first house number range for the match candidate is desired; FALSE if the next range is to retrieved, in terms of sequentially stepping through all the house number ranges for the street.. [Input]

pHouseRange: A pointer to a variable that will receive a pointer to the requested street house number range. The definition of this structure is given later in this document. [Output] The possible values are:

ODD_NUMBERS
EVEN_NUMBERS
ALL_NUMBERS

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_CAND_INDEX	Match candidate index too large.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_END_OF_DATA	No more data available from the requested set.

GEO_ENG_INVALID_ACCESS_ERR	Invalid attempt to access data.
GEO_ENG_NO_DATA_AVAILABLE	Internal database inconsistency.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures and Defines

```
/*
 *  Defined constants for the character array sizes in all the HOUSE_RANGE
structure.
 */
#define MAX_HOUSE_NUMBER_LEN        25
#define MAX_UNIT_TYPE_LEN            5
#define MAX_UNIT_VALUE_LEN           9
#define MAX_POSTAL_CODE_LEN         10
#define MAX_POSTAL_ADDON_CODE_LEN   10
#define MAX_PLACE_NAME_LEN          35
typedef struct
{
  HOUSE_PARITY  houseParity;
  char          fromHouse[MAX_HOUSE_NUMBER_LEN];
  char          toHouse[MAX_HOUSE_NUMBER_LEN];
  char          unitType[MAX_UNIT_TYPE_LEN];
  char          fromUnit[MAX_UNIT_VALUE_LEN];
  char          toUnit[MAX_UNIT_VALUE_LEN];
  char          postalCode[MAX_POSTAL_CODE_LEN];
  char          postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
  char          placeName[MAX_PLACE_NAME_LEN];
  char          lacs;
  char          carrRoute[MAX_CARRIER_ROUTE_LEN];
} HOUSE_RANGE, *pHOUSE_RANGE;
```

GeoEngInit() — Initialize the Geocoding Engine

Purpose

Initialize all subsystems that are required to support the capabilities of the geocoding engine.

Syntax

short **GeoEngInit** (char *pSearchPath,
char *pDbName,
char *pUserDictPath,
short dbInitRequestFlags,
pGEO_ENG_HANDLE pGeoEngHandle);

Parameters

pSearchPath: Directory search path to locate the database and configuration files. This is a string containing directory paths separated by semicolons. [Input]

pDbName: String containing the base name to use in constructing the names of the database and configuration files. [Input]

pUserDictPath: path to the user dictionary files [Input]

dbInitRequestFlags: An integer that indicates which database components to initialize. The value is the bitwise OR of the following defined constants and indicates which database components are requested to be initialized using the specified search path and database name. [Input]

STREET_DB
CENTROID_DB
USER_DB

pGeoEngHandle: A pointer to an opaque handle that will reference a structure used internally by the geocoding engine.

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_DATABASE_ERR	Invalid database contents.
GEO_ENG_BAD_DB_INIT_FLAGS	Illegal flags requesting database initialization.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_FILE_NOT_FOUND_ERR	Required configuration or database file not found.

GEO_ENG_FILE_OPEN_ERR	Error locating or opening a file.
GEO_ENG_FILE_READ_ERR	Error reading from database file.
GEO_ENG_INDEX_OPEN_ERR	Error opening database index.
GEO_ENG_INDEX_REGISTRATION_ERR	Unable to register an index file.
GEO_ENG_MALLOC_ERR	Memory allocation failure.
GEO_ENG_MATCH_INIT_ERR	Low level initialization error from match subsystem.
GEO_ENG_PARSE_INIT_ERR	Low level initialization error from parsing subsystem.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.
GEO_ENG_ZIP_MAS-TER_FILE_NOT_FOUND_ERR	Zipmastr.cdb or Zipmastr.jdx not found.
GEO_ENG_DBQ_MISSING_LICFILE	License file not found.
GEO_ENG_INVALID_SERIAL_NUMBER	Invalid serial number.
GEO_ENG_NADCON_MISSING_FILE	*.las/*.los files not found.

GeoEngKill() — Shut down the Geocoding Engine

Purpose

Clean up after the geocoding engine driver and the subsystems that were also initialized in response to a previous call to **GeoEngInit**.

Syntax

short **GeoEngKill** (pGEO_ENG_HANDLE pGeoEngHandle);

Parameters

pGeoEngHandle: A pointer to an opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**. [Input and Output]

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.

GeoEngIsEval() — Determine if the engine has been initialized for an evaluation copy of the Address Dictionary

Purpose

The purpose of this call is to allow a geocoding application to determine if it is running against an evaluation copy of the address dictionary.

Syntax

short **GeoEngIsEval** (GEO_ENG_HANDLE geoEngHandle, BOOLEAN *bEval);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the bEval: A pointer to a BOOLEAN value which is returned as TRUE if it is an evaluation copy.

Possible returns:

Return	Explanation
SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_EXCEEDED_LIMIT	Exceeded trial license limit.

GeoEngMatchAddress() — Match a user address against the Address Dictionary

Purpose

Accept a user address that is already broken down into street address, city, state, and postal code. Parse this address into a standard form, and then match it against the Address Dictionary. An optional output parameter will allow the user to view the parsed form of the address. For efficient use of the MapMarker subsystems, the input addresses should be sorted on postal code before this function is invoked. The user should not attempt to free the parsed address pointer. The geocoding engine supports access to match candidates for only one input address at a time. This call will destroy all information related to previous calls to this function for different addresses, and it will invalidate the previous values of assigned pointers returned by this function, **GeoEngGetIndexedCandidate**, or **GeoEngGetNextHouseRange**. This function can be used only if STREET_DB is initialized. After this call, geocoding can be done using any of the match candidates via the functions **GeoEngGetCandidateCoords** or **GeoEngGetIndexedCoords**.

If the input address is a post office box or rural route, matching will not be done, but the parsed address will still be returned if requested.

Syntax

short **GeoEngMatchAddress** (GEO_ENG_HANDLE geoEngHandle,
pADDRESS
pPARSED_ADDRESS
pMATCH_STATUS);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pAddress: A pointer to a structure that contains the input. The definition of this structure is given later in this document. [Input]

pParsed_Address: A pointer to a structure that contains the cleaned (parsed) input street address. If the cleaned street address is not desired, pParsed_Address may be NULL. The definition of this structure is given later in this document. [Optional output]

pMatchStatus: A pointer to a variable that will receive the match status, which is an enumeration type defined later in this document. [Output] The possible values are:

SINGLE_MATCH
MULTIPLE_MATCHES
NO_MATCHES
NO_CANDIDATES
SINGLE_INTERSECT_MATCH
MULTIPLE_INTERSECT_MATCH
NO_INTERSECT_MATCHES

NO_INTERSECT_CANDIDATES
POSSIBLE_INTERSECTION

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_DATABASE_ERR	Invalid database contents.
GEO_ENG_BAD_INPUT_ADDRESS	Input address is invalid or missing fields.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_BAD_USER_DATABASE	Invalid user database contents.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_FILE_READ_ERR	Error reading from database file.
GEO_ENG_INDEX_ACCESS_ERR	Error accessing database index.
GEO_ENG_INVALID_ACCESS_ERR	Invalid attempt to access data.
GEO_ENG_MALLOC_ERR	Memory allocation failure.
GEO_ENG_MATCH_ENGINE_ERR	Matching subsystem error.
GEO_ENG_NO_DATA_AVAILABLE	Internal database inconsistency.
GEO_ENG_NO_STREET_ADDRESS	Street address was missing.
GEO_ENG_PARSE_ENGINE_ERR	Parsing subsystem error.
GEO_ENG_PO_BOX_ADDRESS	Parsing only was done.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures and Defines

```
/*
 *  Defined constants for the character array sizes in the ADDRESS
structure.
 */
#define MAX_BUSINESS_LEN               256
#define MAX_STREET_LEN                 256
#define MAX_CITY_LEN                    40
#define MAX_STATE_LEN                   40
#define MAX_POSTAL_CODE_LEN             10
#define MAX_POSTAL_ADDON_CODE_LEN       10
#define MAX_URBANIZATION_LEN            40
typedef struct
{
  char firm[MAX_BUSINESS_LEN];
  char street[MAX_STREET_LEN];
  char city[MAX_CITY_LEN];
  char state[MAX_STATE_LEN];
```

```
  char postalCode[MAX_POSTAL_CODE_LEN];
  char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
  char urbanization[MAX_URBANIZATION_LEN];
} ADDRESS, *pADDRESS;
/*
 *  Defined constants for the character array sizes in the PARSED_ADDRESS
structure.
 */
#define MAX_HOUSE_NUMBER_LEN            25
#define MAX_HOUSE_PREFIX_LEN            11
#define MAX_HOUSE_SEPARATOR_LEN          3
#define MAX_HOUSE_COORD_LEN              8
#define MAX_HOUSE_SUFFIX_LEN            11
#define MAX_STREET_PREDIR_LEN            3
#define MAX_STREET_PRETYPE_LEN          11
#define MAX_STREET_NAME_LEN             29
#define MAX_STREET_POSTTYPE_LEN          5
#define MAX_STREET_POSTDIR_LEN           3
#define MAX_POSTBOX_LEN                  8
#define MAX_UNIT_TYPE_LEN                5
#define MAX_UNIT_VALUE_LEN               9
typedef struct
{
  char housePrefix[MAX_HOUSE_PREFIX_LEN];
  char houseNumber[MAX_HOUSE_NUMBER_LEN];
  char houseSeparator[MAX_HOUSE_SEPARATOR_LEN];
  char houseCoord[MAX_HOUSE_COORD_LEN];
  char houseSuffix[MAX_HOUSE_SUFFIX_LEN];
  char preDir[MAX_STREET_PREDIR_LEN];
  char preType[MAX_STREET_PRETYPE_LEN];
  char name[MAX_STREET_NAME_LEN];
  char postType[MAX_STREET_POSTTYPE_LEN];
  char postDir[MAX_STREET_POSTDIR_LEN];
  char postalBox[MAX_POSTBOX_LEN];       /*no longer used in version 3*/
  char unitType[MAX_UNIT_TYPE_LEN];
  char unitValue[MAX_UNIT_VALUE_LEN];
} PARSED_ADDRESS, *pPARSED_ADDRESS;
/*
 * MATCH_STATUS declaration, as well as the Defined constants for the values
 *   it can be.
 */
typedef short MATCH_STATUS, *pMATCH_STATUS;
#define INTERSECT_BASE             4
#define SINGLE_MATCH               0
#define MULTIPLE_MATCHES           1
#define NO_MATCHES                 2
#define NO_CANDIDATES              3
#define SINGLE_INTERSECT_MATCH    0 + INTERSECT_BASE
#define MULTIPLE_INTERSECT_MATCH  1 + INTERSECT_BASE
#define NO_INTERSECT_MATCHES      2 + INTERSECT_BASE
#define NO_INTERSECT_CANDIDATES   3 + INTERSECT_BASE
#define POSSIBLE_INTERSECTION     4 + INTERSECT_BASE
```

GeoEngMatchFormattedAddress() — Match a broken-down user address against the Address Dictionary

Purpose

Accept a user address that is already broken down into street address and last line components against the Address Dictionary.

Syntax

```
short GeoEngMatchFormattedAddress (GEO_ENG_HANDLE geoEngHandle,
                pPARSED_ADDRESS   pParsedAddress1,
                pPARSED_ADDRESS   pParsedAddress2,
                pLAST_LINE        pLastLine,
                pMATCH_STATUS pMatchResult);
```

Parameters

geoEngHandle: geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pParsedAddress1: A pointer to a structure that contains the first part of the decomposed input address. [Input]

pParsedAddress2: A pointer to a structure that contains the second part of the decomposed input address. This is only relevant for intersection–type addresses.pParsedAddress2 may be NULL. [Optional Input]

pLastLine: A pointer to a structure that contains the decomposed last line components. [Input]

pMatch_Status: A pointer to a variable that will receive the match status which is an enumeration type. [Output]

SINGLE_MATCH

MULTIPLE_MATCHES

NO_MATCHES

NO_CANDIDATES

SINGLE_INTERSECT_MATCH

MULTIPLE_INTERSECT_MATCH

NO_INTERSECT_MATCHES

NO_INTERSECT_CANDIDATES

POSSIBLE_INTERSECTION

Possible returns

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures and Defines

```
/*
 *  Defined constants for the character array sizes in the PARSED_ADDRESS
structure.
 */
#define MAX_HOUSE_NUMBER_LEN            25
#define MAX_HOUSE_PREFIX_LEN            11
#define MAX_HOUSE_SEPARATOR_LEN          3
#define MAX_HOUSE_COORD_LEN              8
#define MAX_HOUSE_SUFFIX_LEN            11
#define MAX_STREET_PREDIR_LEN            3
#define MAX_STREET_PRETYPE_LEN          11
#define MAX_STREET_NAME_LEN             29
#define MAX_STREET_POSTTYPE_LEN          5
#define MAX_STREET_POSTDIR_LEN           3
#define MAX_POSTBOX_LEN                  8
#define MAX_UNIT_TYPE_LEN                5
#define MAX_UNIT_VALUE_LEN               9
typedef struct
{
  char housePrefix[MAX_HOUSE_PREFIX_LEN];
  char houseNumber[MAX_HOUSE_NUMBER_LEN];
  char houseSeparator[MAX_HOUSE_SEPARATOR_LEN];
  char houseCoord[MAX_HOUSE_COORD_LEN];
  char houseSuffix[MAX_HOUSE_SUFFIX_LEN];
  char preDir[MAX_STREET_PREDIR_LEN];
  char preType[MAX_STREET_PRETYPE_LEN];
  char name[MAX_STREET_NAME_LEN];
  char postType[MAX_STREET_POSTTYPE_LEN];
  char postDir[MAX_STREET_POSTDIR_LEN];
  char postalBox[MAX_POSTBOX_LEN];       /*no longer used in version 3*/
  char unitType[MAX_UNIT_TYPE_LEN];
  char unitValue[MAX_UNIT_VALUE_LEN];
} PARSED_ADDRESS, *pPARSED_ADDRESS;
/*
 *  Defined constants for the character array sizes in the LAST_LINE
structure.
 */
#define MAX_CITY_LEN                    40
#define MAX_STATE_LEN                   40
```

```
#define MAX_POSTAL_CODE_LEN             10
#define MAX_POSTAL_ADDON_CODE_LEN       10
typedef struct
{
  char city[MAX_CITY_LEN];
  char state[MAX_STATE_LEN];
  char postalCode[MAX_POSTAL_CODE_LEN];
  char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
} LAST_LINE, *pLAST_LINE;
/*
 * MATCH_STATUS declaration, as well as the Defined constants for the values
 *   it can be.
 */
typedef short MATCH_STATUS, *pMATCH_STATUS;
#define INTERSECT_BASE            4
#define SINGLE_MATCH              0
#define MULTIPLE_MATCHES          1
#define NO_MATCHES                2
#define NO_CANDIDATES             3
#define SINGLE_INTERSECT_MATCH    0 + INTERSECT_BASE
#define MULTIPLE_INTERSECT_MATCH  1 + INTERSECT_BASE
#define NO_INTERSECT_MATCHES      2 + INTERSECT_BASE
#define NO_INTERSECT_CANDIDATES   3 + INTERSECT_BASE
#define POSSIBLE_INTERSECTION     4 + INTERSECT_BASE
```

GeoEngNAD27ToNAD83() — Transforms coordinates from NAD27 to NAD83 Datum

Purpose

Uses the NADCON method to transform coordinate values from the North American Datum of 1927 (NAD 27) to the North American Datum of 1983 (NAD 83).

Syntax

short GeoEngNAD27ToNAD83(GEO_ENG_HANDLE geoEngHandle,
pDOUBLE_POINT pInputPoint,
pDOUBLE_POINT pOutputPoint);

Parameters

geoEngHandle: geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pInputPoint: A pointer to a structure which contains the NAD 27 datum coordinate values. [Input]

pOutputPoint: A pointer to a structure which will receive the NAD 83 datum coordinate values. [Output]

Possible returns

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.

Structures and Defines

```
/*
 *  DOUBLE_POINT structure definition. It is how we return all
 *     coordinates.
 */
typedef struct {
  double x;
  double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
```

GeoEngNAD83ToNAD27() — Transforms coordinates from NAD83 to NAD27 Datum

Purpose

Uses the NADCON method to transform coordinate values from the North American Datum of 1983 (NAD 83) to the North American Datum of 1927 (NAD 27).

Syntax

short GeoEngNAD83ToNAD27(GEO_ENG_HANDLE geoEngHandle,
pDOUBLE_POINT pInputPoint,
pDOUBLE_POINT pOutputPoint);

Parameters

geoEngHandle: geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pInputPoint: A pointer to a structure which contains the NAD 83 datum coordinate values. [Input]

pOutputPoint: A pointer to a structure which will receive the NAD 27 datum coordinate values. [Output]

Possible returns

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.

Structures and Defines

```
/*
 *  DOUBLE_POINT structure definition. It is how we return all
 *     coordinates.
 */
typedef struct {
  double x;
  double y;
} DOUBLE_POINT, *pDOUBLE_POINT;
```

GeoEngParseLastLine() — Parse a last line into city, state and ZIP Code components

Purpose

Accept a last line string and parse it into city, state, zip, and plus4 components. The string passed into this function is preserved (it works with a copy of it).

Syntax

short GeoEngParseLastLine(GEO_ENG_HANDLE geoEngHandle,
char *pInputLastLine,
pLastLine pLastLine);

Parameters

geoEngHandle: geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pInputLastLine: A string which contains the last line to be parsed. [Input]

pLastLine: A pointer to a structure that will receive the decomposed last line components. [Input]

Possible returns

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.

Structures and Defines

```
/*
 *  Defined constants for the character array sizes in the LAST_LINE
structure.
 */
#define MAX_CITY_LEN                    40
#define MAX_STATE_LEN                   40
#define MAX_POSTAL_CODE_LEN             10
#define MAX_POSTAL_ADDON_CODE_LEN       10
typedef struct
{
  char city[MAX_CITY_LEN];
  char state[MAX_STATE_LEN];
  char postalCode[MAX_POSTAL_CODE_LEN];
  char postalAddOnCode[MAX_POSTAL_ADDON_CODE_LEN];
} LAST_LINE, *pLAST_LINE;
```

GeoEngSetCfgParms() — Set the runtime parameters

Purpose

Sets run time parameters for the geocoding engine.

Syntax

short **GeoEngSetCfgParms** (GEO_ENG_HANDLE geoEngHandle,
pGEO_ENG_CFG_PARMS
pGeoEngParms);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

pGeoEngParms: A pointer at a structure containing parameters that affect the behavior of the geocoding engine. The definition of this structure is given later in this document. [Input]

Possible returns:

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_BAD_USER_DATABASE	Invalid user database contents.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.
GEO_ENG_DATABASE_ACCESS_ERR	Error accessing the Address Dictionary.
GEO_ENG_FILE_READ_ERR	Error reading from database file.
GEO_ENG_INDEX_ACCESS_ERR	Error accessing database index.
GEO_ENG_INVALID_ACCESS_ERR	Invalid attempt to access data.
GEO_ENG_MALLOC_ERR	Memory allocation failure.
GEO_ENG_MATCH_ENGINE_ERR	Matching subsystem error.
GEO_ENG_NO_DATA_AVAILABLE	Internal database inconsistency.
GEO_ENG_NO_STREET_ADDRESS	Street address was missing.
GEO_ENG_PARSE_ENGINE_ERR	Parsing subsystem error.
GEO_ENG_PO_BOX_ADDRESS	Parsing only was done.
GEO_ENG_UNINIT_DB_COMPONENT	Requested database component failed to initialize.

Structures

```
*
 * BOOLEAN with TRUE and FALSE.
 */
#define BOOLEAN short
/*
 * DIST_UNIT declaration and all the values it can have.
 */
typedef short DIST_UNIT, *pDIST_UNIT;
#define DIST_UNIT_FOOT          0
#define DIST_UNIT_DEGREE        1
#define DIST_UNIT_INCH          2
#define DIST_UNIT_LINK          3
#define DIST_UNIT_SURVEY_FOOT   4
#define DIST_UNIT_YARD          5
#define DIST_UNIT_ROD           6
#define DIST_UNIT_CHAIN         7
#define DIST_UNIT_MILE          8
#define DIST_UNIT_NAUTICAL_MILE 9
#define DIST_UNIT_MILLIMETER   10
#define DIST_UNIT_CENTIMETER   11
#define DIST_UNIT_METER        12
#define DIST_UNIT_KILOMETER    13
typedef struct
{
  double                linearInset;
  double                perpendicularSetback;
  DIST_UNIT             distUnit;
  BOOLEAN               relaxHouseNumber;
  BOOLEAN               relaxPostalCode;
  BOOLEAN               relaxName;
  BOOLEAN               relaxFinance;
  BOOLEAN               relaxFinanceInState;
  BOOLEAN               matchIntersections;
  BOOLEAN               useCassMode;
  short                 relaxDistance;
} GEO_ENG_CFG_PARMS, *pGEO_ENG_CFG_PARMS;
```

GeoEngSetInteractive() — Set engine to do interactive matching

Purpose

This call is to relax all parameters for the next call to GeoEngMatch Address. The call enables an interactive geocoder to retrieve the maximum number of candidates from the address dictionary. After the next call to GeoEngMatchAddress the flag is automatically reset.

Syntax

short **GeoEngSetInteractive** (GEO_ENG_HANDLE geoEngHandle);

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

Possible returns

SUCCESS	Normal return for the function. It does not necessarily mean there was a match.
GEO_ENG_BAD_PARAM_ERR	NULL input parameter or invalid parameter values.
GEO_ENG_CORRUPTED_ERR	Memory overwrite detected.

MapMarker GeoEngine Error Codes

Error Code	Error #
Error codes shared by multiple components	
#define GEO_ENG_MALLOC_ERR	1
#define GEO_ENG_CORRUPTED_ERR	2
#define GEO_ENG_BAD_PARAM_ERR	3
#define GEO_ENG_FILE_NOT_FOUND_ERR	4
#define GEO_ENG_FILE_OPEN_ERR	5
#define GEO_ENG_FILE_READ_ERR	6
#define GEO_ENG_FILE_WRITE_ERR	7
#define GEO_ENG_INDEX_OPEN_ERR	8
#define GEO_ENG_INDEX_ACCESS_ERR	9
#define GEO_ENG_BAD_DATABASE_ERR	10
#define GEO_ENG_END_OF_DATA	11
#define GEO_ENG_BAD_POSTAL_CODE	12
#define GEO_ENG_UNINIT_DB_COMPONENT	13
#define GEO_ENG_BAD_INPUT_ADDRESS	14
#define GEO_ENG_NO_DATA_AVAILABLE	15
#define GEO_ENG_INVALID_ACCESS_ERR	16
#define GEO_ENG_BAD_LICFILE_ERR	18
#define GEO_ENG_EXCEEDED_LIMIT	19
Controller component error codes	
#define GEO_ENG_COORDS_NOT_AVAILABLE	51
#define GEO_ENG_BAD_CAND_INDEX	52
#define GEO_ENG_BAD_RANGE_INDEX	53
#define GEO_ENG_PO_BOX_ADDRESS	55
#define GEO_ENG_RURAL_RTE_ADDRESS	57
#define GEO_ENG_MULTIPLE_INSTANCE	58
#define GEO_ENG_BAD_INTERSECT_INDEX	59
#define GEO_ENG_BAD_INTERSECT_FORMAT	60
#define GEO_ENG_INTERSECTION_ADDRESS	61

Generic error codes	
Parser component error codes	
#define GEO_ENG_PARSE_INIT_ERR	307
#define GEO_ENG_PARSE_ENGINE_ERR	308
#define GEO_ENG_NO_STREET_ADDRESS	309
#define GEO_ENG_PARSE_STACK_OVERFLOW	310
Matcher component error codes	
#define GEO_ENG_MATCH_INIT_ERR	563
#define GEO_ENG_MATCH_ENGINE_ERR	564
Interpreter component error codes	
#define GEO_ENG_INTERP_HOUSE_ERR	819
#define GEO_ENG_INTERP_BAD_INSET	820
#define GEO_ENG_INTERP_BAD_SETBACK	821
ZIP Centroid file not found	
#define GEO_ENG_Z9_CEN_FILE_NOT_FOUND_ERR	1045
Database component error codes	
#define GEO_ENG_DATABASE_ACCESS_ERR	1075
#define GEO_ENG_BAD_DB_INIT_FLAGS	1076
#define GEO_ENG_UNDEFINED_SEARCH	1077
#define GEO_ENG_MISSING_DATABASE_ERR	1078
#define GEO_ENG_DBQ_MISSING_LICFILE	1079
#define GEO_ENG_DBQ_MISSING_FILEID_TABLE	1080
#define GEO_ENG_DBQ_BAD_HEADER	1081
#define GEO_ENG_DBQ_BAD_INDEX	1082
#define GEO_ENG_BETA_EXPIRATION_ERR	1084
#define GEO_ENG_INVALID_SERIAL_NUMBER	1085
#define GEO_ENG_DBQ_UDR_ADR_CONFLICT_ERR	1086
#define GEO_ENG_DBQ_USER_DICT_INIT_ERR1087	
ZIP master file errors	
#define GEO_ENG_ZIP_MASTER_FILE_NOT_FOUND_ERR	1228
#define GEO_ENG_ZIP_MASTER_FILE_READ_ERR	1230
#define GEO_ENG_ZIP_MASTER_INDEX_OPEN_ERR	1232
#define GEO_ENG_ZIP_MASTER_INDEX_ACCESS_ERR	1233

NADCON datum conversion file errors	
#define GEO_ENG_NADCON_MISSING_FILE	1334
#define GEO_ENG_DD_ALLOCATE_ERR	700
#define GEO_ENG_DD_FIELD_DOES_NOT_EXIST_ERR	701
#define GEO_ENG_DD_END_ROW_TOO_LARGE_ERR	702
#define GEO_ENG_DD_START_ROW_GT_END_ROW_ERR	703
#define GEO_ENG_DD_START_ROW_NEGATIVE_ERR	704
#define GEO_ENG_DD_INVALID_FIELD_NAME_ERR	705
#define GEO_ENG_DD_INVALID_UDINFO_ERR	706
#define GEO_ENG_DD_COPY_FIELD_ERR	707
#define GEO_ENG_DD_TEMP_DIR_ERR	708
#define GEO_ENG_DD_ADD_PATH_SLASH_ERR	709
#define GEO_ENG_DD_GET_FIELD_ERR	710
#define GEO_ENG_DD_REMOVE_FILE_ERR	711
#define GEO_ENG_DD_BAD_HANDLE_ERR	712
#define GEO_ENG_DD_INVALID_OBJECT_ERR	713
#define GEO_ENG_DD_GET_CWD_ERR	714
#define GEO_ENG_DD_SET_CWD_ERR	715
#define GEO_ENG_DD_BAD_PARAM_ERR	716
#define GEO_ENG_DD_NAME_TOO_LONG_ERR	717
#define GEO_ENG_DD_INVALID_FILE_EXT_ERR	718
#define GEO_ENG_DD_INVALID_PROCESS_CODE_ERR	719
#define GEO_ENG_DD_TRANSLATE_TABLE_ERR	720
#define GEO_ENG_DD_CLOSE_ALL_ERR	721
#define GEO_ENG_DD_RENAME_FILE_ERR	722
#define GEO_ENG_DD_DIR_DOES_NOT_EXIST_ERR	723
#define GEO_ENG_DD_INVALID_PROJECTION_ERR	724
#define GEO_ENG_DD_INVALID_DATUM_ERR	725
#define GEO_ENG_DD_BAD_PARAMETER_ERR	726
#define GEO_ENG_DD_INVALID_DICT_NAME_ERR	727
#define GEO_ENG_DD_EMPTY_FIELD_ERR	728
#define GEO_ENG_DD_DICT_NAME_TOO_LONG_ERR	729
#define GEO_ENG_DD_DATUM_CONVERSION_ERR	730
#define GEO_ENG_DD_INPUT_OUTPUT_DIR_SAME_ERR	731
#define GEO_ENG_DD_CREATE_LOG_FILE_ERR	732

#define GEO_ENG_DD_CLOSE_LOG_FILE_ERR	733
#define GEO_ENG_DD_WRITE_LOG_FILE_ERR	734
MI based error codes	
#define GEO_ENG_DD_MI_OPEN_TABLE_ERR	800
#define GEO_ENG_DD_MI_GET_NUM_ROWS_ERR	801
#define GEO_ENG_DD_MI_GET_NUM_FIELDS_ERR	802
#define GEO_ENG_DD_MI_GET_ATTR_DEFS_ERR	803
#define GEO_ENG_DD_MI_CLOSE_TABLE_ERR	804
#define GEO_ENG_DD_MI_FETCH_ROW_ERR	805
#define GEO_ENG_DD_MI_INIT_ERR	806
#define GEO_ENG_DD_MI_TERM_ERR	807
#define GEO_ENG_DD_MI_PREPARE_TABLE_ERR	808
#define GEO_ENG_DD_MI_GET_COORDSYS_ERR	809
Database error codes	
#define GEO_ENG_DD_CREATE_STREET_DBF_ERR	900
#define GEO_ENG_DD_OPEN_STREET_DBF_ERR	901
#define GEO_ENG_DD_CLOSE_STREET_DBF_ERR	902
#define GEO_ENG_DD_INIT_STREET_DBF_ERR	903
#define GEO_ENG_DD_CREATE_SEGMENT_DBF_ERR	904
#define GEO_ENG_DD_OPEN_SEGMENT_DBF_ERR	905
#define GEO_ENG_DD_CLOSE_SEGMENT_DBF_ERR	906
#define GEO_ENG_DD_INIT_SEGMENT_DBF_ERR	907
#define GEO_ENG_DD_CREATE_RANGE_DBF_ERR	908
#define GEO_ENG_DD_OPEN_RANGE_DBF_ERR	909
#define GEO_ENG_DD_CLOSE_RANGE_DBF_ERR	910
#define GEO_ENG_DD_INIT_RANGE_DBF_ERR	911
#define GEO_ENG_DD_CREATE_POINT_DBF_ERR	912
#define GEO_ENG_DD_OPEN_POINT_DBF_ERR	913
#define GEO_ENG_DD_CLOSE_POINT_DBF_ERR	914
#define GEO_ENG_DD_INIT_POINT_DBF_ERR	915
#define GEO_ENG_DD_CREATE_STREET2_DBF_ERR	916
#define GEO_ENG_DD_OPEN_STREET2_DBF_ERR	917
#define GEO_ENG_DD_CLOSE_STREET2_DBF_ERR	918
#define GEO_ENG_DD_INIT_STREET2_DBF_ERR	919
#define GEO_ENG_DD_CREATE_SEGMENT2_DBF_ERR	920

```
#define GEO_ENG_DD_OPEN_SEGMENT2_DBF_ERR 921
#define GEO_ENG_DD_CLOSE_SEGMENT2_DBF_ERR 922
#define GEO_ENG_DD_INIT_SEGMENT2_DBF_ERR 923
#define GEO_ENG_DD_CREATE_RANGE2_DBF_ERR 924
#define GEO_ENG_DD_OPEN_RANGE2_DBF_ERR 925
#define GEO_ENG_DD_CLOSE_RANGE2_DBF_ERR 926
#define GEO_ENG_DD_INIT_RANGE2_DBF_ERR 927
#define GEO_ENG_DD_CREATE_POINT2_DBF_ERR 928
#define GEO_ENG_DD_OPEN_POINT2_DBF_ERR 929
#define GEO_ENG_DD_CLOSE_POINT2_DBF_ERR 930
#define GEO_ENG_DD_INIT_POINT2_DBF_ERR 931
#define GEO_ENG_DD_WRITE_STREET2_DBF_ERR 932
#define GEO_ENG_DD_WRITE_SEGMENT2_DBF_ERR 933
#define GEO_ENG_DD_WRITE_RANGE2_DBF_ERR 934
#define GEO_ENG_DD_WRITE_POINT2_DBF_ERR 935
#define GEO_ENG_DD_COPY_STREET_REC_ERR 936
#define GEO_ENG_DD_COPY_SEGMENT_REC_ERR 937
#define GEO_ENG_DD_COPY_RANGE_REC_ERR 938
#define GEO_ENG_DD_COPY_POINT_REC_ERR 939
#define GEO_ENG_DD_CREATE_ZIPCODE_DBF_ERR 940
#define GEO_ENG_DD_CLOSE_ZIPCODE_DBF_ERR 941
#define GEO_ENG_DD_CREATE_STREET_DIR_DBF_ERR 942
#define GEO_ENG_DD_CLOSE_STREET_DIR_DBF_ERR 943
#define GEO_ENG_DD_CREATE_STREET_NAME_DBF_ERR 944
#define GEO_ENG_DD_CLOSE_STREET_NAME_DBF_ERR 945
#define GEO_ENG_DD_CREATE_STREET_TYPE_DBF_ERR 946
#define GEO_ENG_DD_CLOSE_STREET_TYPE_DBF_ERR 947
#define GEO_ENG_DD_CREATE_CITY_DBF_ERR 948
#define GEO_ENG_DD_OPEN_CITY_DBF_ERR 949
#define GEO_ENG_DD_CLOSE_CITY_DBF_ERR 950
#define GEO_ENG_DD_INIT_CITY_DBF_ERR 951
#define GEO_ENG_DD_CREATE_ZIP5INFO_DBF_ERR 952
#define GEO_ENG_DD_OPEN_ZIP5INFO_DBF_ERR 953
#define GEO_ENG_DD_CLOSE_ZIP5INFO_DBF_ERR 954
#define GEO_ENG_DD_INIT_ZIP5INFO_DBF_ERR 955
```

#define GEO_ENG_DD_WRITE_STREET_DBF_ERR	956
#define GEO_ENG_DD_WRITE_SEGMENT_DBF_ERR	957
#define GEO_ENG_DD_WRITE_RANGE_DBF_ERR	958
#define GEO_ENG_DD_WRITE_POINT_DBF_ERR	959
#define GEO_ENG_DD_WRITE_ZIP5INFO_DBF_ERR	960
#define GEO_ENG_DD_WRITE_CITYFIN_DBF_ERR	961
#define GEO_ENG_DD_WRITE_ADR_FILE_ERR	962
#define GEO_ENG_DD_CREATE_USAFIN_DBF_ERR	963
#define GEO_ENG_DD_WRITE_FILEID_FILE_ERR	964
#define GEO_ENG_DD_CREATE_UFD_FILE_ERR	965
#define GEO_ENG_DD_OPEN_FINANCE_FILE_ERR	966
#define GEO_ENG_DD_CLOSE_FINANCE_FILE_ERR	967
#define GEO_ENG_DD_WRITE_UFD_HEADER_ERR	968
#define GEO_ENG_DD_WRITE_UFD_FILE_ERR	969
#define GEO_ENG_DD_ZIP_MASTER_OPEN_ERR	970
#define GEO_ENG_DD_CREATE_ZIP_MASTER_FILE_ERR	971
#define GEO_ENG_DD_CREATE_ZIP_MASTER_INDEX_ERR	972
#define GEO_ENG_DD_CREATE_TEMP_FILE_ERR	973
#define GEO_ENG_DD_WRITE_ZIP_MASTER_HEADER_ERR	974
#define GEO_ENG_DD_ADD_ZIP_MASTER_INDEX_ERR	975
#define GEO_ENG_DD_ADD_ZIP_MASTER_POS_ERR	976
#define GEO_ENG_DD_ADD_ZIP_MASTER_WRITE_ERR	977
#define GEO_ENG_DD_ZIP_MASTER_ERR	978
#define GEO_ENG_DD_ADD_TEMP_INDEX_ERR	979
#define GEO_ENG_DD_CREATE_FINANCE_CENT_FILE_ERR	980
#define GEO_ENG_DD_CLOSE_ADR_FILE_ERR	981
#define GEO_ENG_DD_CLOSE_ADX_FILE_ERR	982
#define GEO_ENG_DD_CITY_INDEX_ERR	983
Cancel Code	
#define GEO_ENG_DD_CANCEL_DETECTED	999

Programming Usage Example

```
/*
 *  GEOSAMP - Sample application demonstrating the use of the
MapInfo Geocoding Engine
 */
/* Include the standard C things */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
/* For WINDOWS include these */
#if defined(_WINDOWS)||defined(_WIN32)
  #include <windows.h>
#endif
/* Include the files needed for the Geocoding Engine. */
#ifdef _WINDOWS
  #define GEODLL
#endif
#include <geo.h>
#include <geoerror.h>
/* Prototypes for local functions */
short InitializeEngine(pGEO_ENG_HANDLE);
short SingleMatch (GEO_ENG_HANDLE, pADDRESS);
short MultiMatch (GEO_ENG_HANDLE, pADDRESS);
short NoMatch (GEO_ENG_HANDLE, pADDRESS);
void  PrintStreetCandidate (pCANDIDATE_ADDRESS, pDOUBLE_POINT);
void  PrintPlaceCandidate (pCANDIDATE_ADDRESS, pDOUBLE_POINT);
void  PrintInputAddress(pADDRESS);
short GetInputAddress(pADDRESS);
short FindCentroid(GEO_ENG_HANDLE, pADDRESS);
void  PrintHouseRange(pHOUSE_RANGE);
short NoCandidates(GEO_ENG_HANDLE, pADDRESS, char *);
/*
 * Define FAILURE to not be SUCCESS
 */
#define FAILURE (!SUCCESS)
void main (void)
{
   short             matchStatus;
   GEO_ENG_HANDLE      geoEngHandle;
   ADDRESS     inputAddress;
   PARSED_ADDRESS    cleanAddress;
    short             retCode;

    /*
```

```
     * Initialize the Geocoding Engine. NOTE: If initialization
fails, then we do not
     *   have any cleanup to do, so just get out.
     */
   retCode = InitializeEngine(&geoEngHandle);

   if (retCode != SUCCESS)
    {
        exit(1);
    }
   /*
    * Geocode some addresses.
    */
    /* initialize retCode so we pass the test the first time */
    retCode = SUCCESS;
    while (retCode == SUCCESS && GetInputAddress(&inputAddress) ==
SUCCESS)
    {

      retCode = GeoEngMatchAddress(geoEngHandle,&inputAddress,
                                   &cleanAddress,&matchStatus);
      if (retCode == GEO_ENG_PO_BOX_ADDRESS)
        {
         fprintf (stderr, "Input Address was a PO Box\n", retCode);
            retCode = FindCentroid(geoEngHandle, &inputAddress);
        } else if (retCode == GEO_ENG_RURAL_RTE_ADDRESS)
        {
         fprintf (stderr, "Input Address was a PO Box\n", retCode);
            retCode = FindCentroid(geoEngHandle, &inputAddress);
        }
      else if (retCode == SUCCESS)
      {
         switch (matchStatus)
         {
            case SINGLE_MATCH:
               retCode = SingleMatch(geoEngHandle,&inputAddress);
               break;
            case MULTIPLE_MATCHES:
               retCode = MultiMatch(geoEngHandle,&inputAddress);
                  break;
            case NO_MATCHES:
               retCode = NoMatch(geoEngHandle,&inputAddress);
                  break;
            case NO_CANDIDATES:
                    retCode = NoCandidates(geoEngHandle,
&inputAddress,
```

```
                                                cleanAddress.name);
                    break;
               default:
                  printf("Invalid match status = %d\n",matchStatus);
                    break;
          }

        }
        else
        {
            fprintf(stderr, "Error %d from GeoEngMatchAddress\n");
            continue;
      }
    } /* end while GetInputAddress */
    printf("Killing the Geocoding Engine\n");
   retCode = GeoEngKill(&geoEngHandle);
   if (retCode != SUCCESS)
    {
      fprintf (stderr, "Error %d from GeoEngKill shut down\n",
retCode);
        exit(1);
    }

    printf("\nThanks for using the MapInfo Geocoding Engine Sample
Application!\n");
   exit (0);
} /* end main */
/*
 * InitializeEngine - Fires up the MapInfo Geocoding Engine. For
Windows it
 *   consults the user ini file. For other platforms it asks for
input.
 */
short InitializeEngine(pGEO_ENG_HANDLE pGeoEngHandle)
{
   char dbPath[256]   = "";
   char *dbName    = "geo_usa";
   GEO_ENG_CFG_PARMS geoEngParms;
    short retCode;
    short dbFlags;

   printf("Initializing the Geocoding Engine. This could take
awhile.\n");
#if 0
    /* get the database path and database name from MAPINFO.INI */
```

```
    GetPrivateProfileString("SYSTEM","DatabasePath","
",dbPath,sizeof(dbPath),
                            "MAPINFO.INI");
#else /* if not in Windows, use standard input to get the database
path and name */
   printf("Please enter the full path of the MapMarker address
dictionary\n");
   gets (dbPath);
#endif

    /*
     * Check availability, we only initialize if STREET_DB and
CENTROID_DB
     *    are available.
     */
    retCode = GeoEngCheckDbAvailability (dbPath, dbName, NULL,
&dbFlags);
   if (retCode != SUCCESS)
   {
      fprintf (stderr, "Error %d checking MapMarker dictionary
availability\n",retCode);
      return FAILURE;
   }
    if (!(dbFlags & STREET_DB))
    {
        fprintf(stderr, "Address Dictionary Street files not
available\n");
        fprintf(stderr, "Please try again with the correct
dbPath\n");
      return FAILURE;
    }
    if (!(dbFlags & CENTROID_DB))
    {
        fprintf(stderr, "Address Dictionary Centroid files not
available\n");
        fprintf(stderr, "Please try again with the correct
dbPath\n");
      return FAILURE;
    }
    /*
     * If I got here, then try and initialize the engine.
     */
   retCode = GeoEngInit(dbPath, dbName, NULL,
                        STREET_DB | CENTROID_DB, pGeoEngHandle);
   if (retCode != SUCCESS)
   {
```

```
      fprintf (stderr, "Error %d initializing MapMarker\n",retCode);
      return FAILURE;
   }
   else
   {
       /*
         * Get the current configuration values and set them to what
we want.
         */
      retCode = GeoEngGetCfgParms (*pGeoEngHandle, &geoEngParms);
      if (retCode != SUCCESS)
        {
         fprintf (stderr, "Error %d reading configuration
parameters\n",retCode);
          return FAILURE;
        }
       else
      {
         geoEngParms.relaxHouseNumber = FALSE;
         geoEngParms.relaxName = TRUE;
         geoEngParms.relaxPostalCode = TRUE;
            retCode = GeoEngSetCfgParms(*pGeoEngHandle,
&geoEngParms);
         if (retCode != SUCCESS)
            {
            fprintf (stderr, "Warning: Error %d received setting
configuration parameters\n",retCode);
               fprintf (stderr, "\tDEFAULT VALUES FOR ALL
CONFIGURATION PARAMETERS WILL BE USED\n");
            }
        }
   }
    return SUCCESS;
} /* end InitializeEngine function */
/*
 * GetInputAddress - prompts a user for the address fields, and
fills in an
 *   ADDRESS structure.
 */
short GetInputAddress(pADDRESS pInputAddress)
{
    char tmpBuf[256];
    if (!pInputAddress)
        return FAILURE;
    printf("\nWhen prompted, please enter a street address, city,
state and ZIP Code:\n");
```

```
    printf("Street Address (Enter to quit) = ");
    gets (tmpBuf);
    if (!strlen(tmpBuf))
        return FAILURE;
    strcpy(pInputAddress->street, tmpBuf);
    printf("City = ");
    gets (tmpBuf);
    strcpy(pInputAddress->city, tmpBuf);
    printf("State = ");
    gets (tmpBuf);
    strcpy(pInputAddress->state, tmpBuf);
    printf("Five or Nine Digit ZIP Code = ");
    gets (tmpBuf);
    if (strlen(tmpBuf) > (size_t)5)
    {
      int i;
      /* Had a plus 4 */
      i = 5;
      if (!isdigit(tmpBuf[5]))
      {
         i++;
      }
      strcpy(pInputAddress->postalAddOnCode, tmpBuf);
      tmpBuf[5] = 0;
    }
      strcpy(pInputAddress->postalCode, tmpBuf);
      return SUCCESS;
}
/*
 * SingleMatch prints out the matching candidate
 */
short SingleMatch(GEO_ENG_HANDLE geoEngHandle,
                  pADDRESS pInputAddress)
{
    CANDIDATE_ADDRESS matchAddress;
    DOUBLE_POINT      location;
    short retCode;
    short precision;
    printf("A single good match for the address was found.\n");
    PrintInputAddress(pInputAddress);
   retCode =
GeoEngGetIndexedCandidate(geoEngHandle,0,&matchAddress);
   if (retCode != SUCCESS)
    {
      fprintf (stderr, "Error %d getting candidate
address\n",retCode);
```

```
        return retCode;
     }
   else
   {
      retCode = GeoEngGetCandidateCoords(geoEngHandle,0,
                                         &precision, &location);
      if (retCode != SUCCESS)
        {
         fprintf (stderr, "Error %d getting candidate
coordinates\n",retCode);
            return retCode;
        }
      else
        {
            if (matchAddress.addressType == ADDRESS_TYPE_STREET)
            PrintStreetCandidate(&matchAddress, &location);
            else
            PrintPlaceCandidate(&matchAddress, &location);
        }
   }
   return SUCCESS;
} /* end oneMatch function */
/*
 * MultiMatch - For each of the match candidates we will print out
the candidate.
 */
short MultiMatch(GEO_ENG_HANDLE geoEngHandle,
                 pADDRESS pInputAddress)
{
   unsigned short    i;
    unsigned short closeMatchCount;
    unsigned short candidateCount;
    CANDIDATE_ADDRESS matchAddress;
    short retCode;
    short precision;
    DOUBLE_POINT location;

   printf("There were multiple matches for this address\n");
    PrintInputAddress(pInputAddress);
   retCode = GeoEngGetCandidateCount(geoEngHandle,&closeMatchCount,
                                     &candidateCount);
   if (retCode != SUCCESS)
    {
      fprintf (stderr, "Error %d getting candidate
count\n",retCode);
        return retCode;
```

```
    }
   else if (closeMatchCount < 2)
    {
        fprintf(stderr, "Error: CloseMatchCount %d in multiMatch\n",
closeMatchCount);
        return FAILURE;
    }
    else
    {
        for (i = 0; i < closeMatchCount; i++)
        {
         retCode = GeoEngGetIndexedCandidate(geoEngHandle, i,
&matchAddress);
         if (retCode != SUCCESS)
            {
            fprintf (stderr, "Error %d getting match
candidate\n",retCode);
                return retCode;
            }
         else
         {
            retCode = GeoEngGetCandidateCoords(geoEngHandle, i,
                                               &precision,
&location);
            if (retCode != SUCCESS)
                {
               fprintf (stderr, "Error %d getting close match
coordinates\n",retCode);
                    return retCode;
                }
            else
                {
                    if (matchAddress.addressType ==
ADDRESS_TYPE_STREET)
                     PrintStreetCandidate(&matchAddress, &location);
                    else
                     PrintPlaceCandidate( &matchAddress, &location);
                }
         }
        }
    }
   return SUCCESS;
} /* end MultiMatch function */
/*
 * NoMatch - For each candidate I dump out all the address ranges
for the candidate.
```

```
 */
short NoMatch(GEO_ENG_HANDLE geoEngHandle,
              pADDRESS pInputAddress)
{
   unsigned short    i;
    unsigned short closeMatchCount;
    unsigned short candidateCount;
    CANDIDATE_ADDRESS matchAddress;
    short retCode;
    short precision;
    DOUBLE_POINT location;
    BOOLEAN bFirst;
    HOUSE_RANGE houseRange;
    printf("No match for the given address\n");
    PrintInputAddress(pInputAddress);

   retCode = GeoEngGetCandidateCount(geoEngHandle,&closeMatchCount,
                                     &candidateCount);
   if (retCode != SUCCESS)
    {
      fprintf (stderr, "Error %d getting candidate
count\n",retCode);
        return retCode;
    }
   else if (closeMatchCount > 0)
    {
        fprintf(stderr, "Error: CloseMatchCount %d in NoMatch\n",
closeMatchCount);
        return FAILURE;
    }
    else
    {
        for (i = 0; i < candidateCount; i++)
        {
         retCode = GeoEngGetIndexedCandidate(geoEngHandle, i,
&matchAddress);
         if (retCode != SUCCESS)
            {
            fprintf (stderr, "Error %d getting match
candidate\n",retCode);
                return retCode;
            }
         else
         {
            retCode = GeoEngGetCandidateCoords(geoEngHandle, i,
```

```
                                                    &precision,
&location);
            if (retCode != SUCCESS)
                {
               fprintf (stderr, "Error %d getting close match
coordinates\n",retCode);
                    return retCode;
                }
            else
                {
                    if (matchAddress.addressType ==
ADDRESS_TYPE_STREET)
                     PrintStreetCandidate(&matchAddress, &location);
                    else
                     PrintPlaceCandidate(&matchAddress, &location);
                    /*
                     * Now dump all the house ranges for the
candidate
                     */
                    printf("\tHouse Ranges\n");
                    bFirst = TRUE;
                    while ((retCode =
GeoEngGetNextHouseRange(geoEngHandle, i,
                                                          bFirst,
&houseRange)) == SUCCESS)
                    {
                        bFirst = FALSE;
                        PrintHouseRange(&houseRange);
                    }
                    if (retCode != GEO_ENG_END_OF_DATA)
                    {
                        fprintf(stderr, "Error %d from
GeoEngGetNextHouseRange\n",
                                retCode);
                        return retCode;
                    }
                }
         }
        }
    }
    return SUCCESS;
}
/*
 * PrintHouseRange - prints out the house range in a nice format.
 */
void PrintHouseRange(pHOUSE_RANGE pHouseRange)
```

```
{
    printf("\tFrom %10.10s To %10.10s ZIP Code %s-%s\n",
            pHouseRange->fromHouse, pHouseRange->toHouse,
            pHouseRange->postalCode, pHouseRange->postalAddOnCode);
}
/*
 *
 */
short FindCentroid(GEO_ENG_HANDLE geoEngHandle,
                   pADDRESS pInputAddress)
 {
   DOUBLE_POINT location;
   short retCode;
   short precision;
   retCode =
GeoEngFindPostalCentroid(geoEngHandle,pInputAddress->postalCode,
                            pInputAddress->postalAddOnCode,
                            &precision, &location, NULL);
   if (retCode != SUCCESS)
    {
      fprintf (stderr, "Error %d getting postal
centroid\n",retCode);
        return retCode;
    }
   else
    {
      printf ("MapMarker Point = %f longitude, %f latitude\n",
location.x,
              location.y);
    }
   return SUCCESS;
} /* end FindCentroid function */
/*
 * PrintStreetCandidate - Prints out the street candidate, and it's
coordinates
 */
void PrintStreetCandidate(pCANDIDATE_ADDRESS pMatchAddress,
                     pDOUBLE_POINT      pLocation)
{
    /*
     * Print the fields.
     */
   printf ("\nCandidate Address\n\t");
   if (strlen(pMatchAddress->candidateAddress.street))
        printf("%s ", pMatchAddress->candidateAddress.street);
    printf("\n\t%s\n", pMatchAddress->lastLine);
```

```
   printf ("MapMarker Point = %f longitude, %f latitude\n\n",
               pLocation->x,
              pLocation->y);
} /* end PrintStreetCandidate function */
/*
 * PrintPlaceCandidate - Prints out the place candidate, and it's
coordinates
 */
void PrintPlaceCandidate(pCANDIDATE_ADDRESS pMatchAddress,
                  pDOUBLE_POINT      pLocation)
{
   printf ("\tCandidate Address\n\t%s\n",
         pMatchAddress->candidateAddress.firm);
    printf("\t%s\n", pMatchAddress->lastLine);
   printf ("MapMarker Point = %f longitude, %f latitude\n\n",
               pLocation->x,
              pLocation->y);

} /* end PrintPlaceCandidate function */
void PrintInputAddress(pADDRESS pInputAddress)
{
    printf("Input Address\n");
    printf("%s\n%s, %s %s %s\n", pInputAddress->street,
           pInputAddress->city, pInputAddress->state,
pInputAddress->postalCode,
           pInputAddress->postalAddOnCode);
}
/*
 * NoCandidates - tries to browse the street names which begin with
the names
 *   first letter.
 */
short NoCandidates(GEO_ENG_HANDLE geoEngHandle,
                   pADDRESS pInputAddress,
                   char *name)
{
    BOOLEAN bDone = FALSE;
    BOOLEAN bFirst = TRUE;
    short   retCode;
    char browseFilter[256];
    char postalCode[256];
    BROWSED_STREET browsedStreet;

    printf("No candidate were found for the given address\n");
    PrintInputAddress(pInputAddress);
    if (strlen(pInputAddress->postalCode) < (size_t)5)
```

```
    {
        printf("Input Address ZIP Code was bad, please enter a new
one\n");
        gets(postalCode);
    }
    else
    {
        strcpy(postalCode, pInputAddress->postalCode);
    }

    while (!bDone)
    {
        if (bFirst && isalpha(name[0]))
        {
            browseFilter[0] = name[0];
            browseFilter[1] = 0;
        }
        else
        {
            printf("Enter a browsing filter (Enter to quit)?\n");
            gets(browseFilter);
        }
        if (!strlen(browseFilter))
        {
            bDone = TRUE;
        }
        else
        {
            bFirst = FALSE;
            retCode = GeoEngBrowseSetFilter(geoEngHandle,
postalCode, browseFilter);
            if (retCode != SUCCESS)
            {
                fprintf(stderr, "Error %d from
GeoEngBrowseSetFilter\n");
                return retCode;
            }
            while((retCode = GeoEngBrowseStreet(geoEngHandle,
&browsedStreet)) == SUCCESS)
            {
                printf("\tStreet %s %s %s %s %s\n",
browsedStreet.preDir,
                        browsedStreet.preType, browsedStreet.name,
                        browsedStreet.postType,
browsedStreet.postDir);
            }
```

```
            if (retCode != GEO_ENG_END_OF_DATA)
            {
                fprintf(stderr, "Error %d from
GeoEngBrowseStreet\n");
                return retCode;
            }
        }
    }
}
```

Chapter 2: Data Dictionary API

Overview

This chapter describes the Data Dictionary API(DD API), a programming interface that you, as a developer, can use to create a customized address dictionary within your application. This API supports the creation of user dictionaries from a MapInfo table. This API works in concert with the MapMarker Geoengine API, discussed in the previous chapter.

In This Chapter . . .

Required and Optional Fields

You must specify the field names in the MapInfo table in order for the table to be translated into a user dictionary. Certain fields are required and must be present in the MapInfo table, while others are optional. If any of the required fields are missing, a missing field error code will be returned to the user. Below are the required and optional fields. These fields represent one object in the MapInfo table.

MapInfo Object (line, polyline or point)	Required Fields	Optional Fields
	Left Start Address	Left ZIP+4 code
	Right Start Address	Right ZIP+4 code
	Left End Address	Left Census Block
	Right End Address	Right Census Block
	Street Name	Left Odd/Even indicator
	State Abbreviation	Right Odd/Even indicator
	Left and Right ZIP Code	Place Name
		City Name

User Dictionary File Format

The user dictionaries that are produced by the DD API are in the same format as the MapMarker Address Dictionary (adr/adx files). The main distinction between the two is that the associated files are named differently. When creating a user dictionary, you must specify a base name of eight characters or less. The name cannot be a duplicate of any MapMarker Address Dictionary file names (i.e., ny12). Also, each user dictionary will contain its own zipmastr, cityfin,usafin and relaxed finance index files. These files are used to look up ZIP Codes or city/state names to find out where to look in the user dictionary file when geocoding. Below is a table that compares the MapMarker Address Dictionary file extensions to a custom user dictionary file set.

MapMarker Address Dictionary Naming Scheme	User Dictionary Naming Scheme
NY12.ADR	STREETS.UDR
NY12.ADX	STREETS.UDX
ZIPMASTR.CDB	STREETS.ZMD
ZIPMASTR.JDX	STREETS.ZMX
CITYFIN.JDX	STREETS.CFX
USAFIN.CDB	STREETS.UFD
FINCTRD.IDX	STREETS.RFX

Data Dictionary Interface

The DD API is written at a very high level so that not much user intervention is required once the initial API call has been made. All of the code for the API is integrated into the GeoEngine of MapMarker. The following defines the function for the Data Dictionary interface.

GeoEngDDCreateUserDictionary

Purpose

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

Syntax

long **GeoEngDDCreateUserDictionary(GEO_ENG_HANDLE** geoEngHandle,
char *input_table_name,
char *output_dictionary_name,
pUD_INFO pUDInfo
(long (*pStatusFunction)(long)))

Parameters

geoEngHandle: An opaque handle that references a structure used internally by the geocoding engine, and which was defined by a previous call to **GeoEngInit**.

input_table_name	Full path of the MapInfo input table name from which a user dictionary is created.
output_dictionary_name	Full path to the output dictionary including the user dictionary file extension .udr.
pUDInfo	InfoPointer to the UD_Info (user dictionary information structure)
pStatusFunction	Pointer to status function

This API routine creates a user dictionary from a MapInfo input table. The user dictionary will be named according to the output_dictionary_name argument given to the routine. The

GeoEngine handle is used to look up ZIP Codes in the zipmastr file and to look up city names in the cityfin file to find corresponding finance numbers. The pointer to the user dictionary info structure must be initialized by the user. This structure must be filled with the table field names that are to be used for translation, some of which are required to be present in the MapInfo source table. An error code will be returned if field names are not found for these fields in the UD_Info structure. The process flag in UD_Info should be set to ALL_ROWS or SUBSET_ROWS with iStartRow and iEndRow set accordingly.

The status function pointer represents a function that can be called to update the status of the API. It gives the end user the opportunity to cancel the dictionary creation process, if she chooses. The status function must take a long integer and return a long integer. It must return a flag which represents whether or not to cancel the operation. By returning 0, the API will continue. However, if 1 is returned then the API will halt as if cancelled. The input parameter will be a number between 1 and 100 which represents the status of the API in percentage. This value can be used to print the status to the screen.

You also have the option of not using the status function. If the status function pointer is NULL or 0, then the API will not attempt to call the status function

Possible returns: see list beginning on Page 55.

Structures and Defines

The user dictionary creation interface needs to know the field names in the MapInfo table and the starting and ending rows for translation. This information is stored in the Data Dictionary data structure. The DD API uses a single large structure to store all of the options, including the names of the columns, the starting and ending row numbers and a process flag that indicates how much of the rows should be processed. Below is the structure definition:

```
#define    MAX_FIELD_NAME_LEN          31
#define    ALL_ROWS                    1
#define    SUBSET_ROWS                 2
typedef struct ud_info
{
char LeftStartAddrFieldName[MAX_FIELD_NAME_LEN + 1];
char LeftEndAddrFieldName[MAX_FIELD_NAME_LEN + 1];
char RightStartAddrFieldName[MAX_FIELD_NAME_LEN + 1];
char RightEndAddrFieldName[MAX_FIELD_NAME_LEN + 1];
char StreetFieldName[MAX_FIELD_NAME_LEN + 1];
char CityFieldName[MAX_FIELD_NAME_LEN + 1];
char StateFieldName[MAX_FIELD_NAME_LEN + 1];
char LeftZipCodeFieldName[MAX_FIELD_NAME_LEN + 1];
char RightZipCodeFieldName[MAX_FIELD_NAME_LEN + 1];
```

```
char LeftZipAddOnFieldName[MAX_FIELD_NAME_LEN + 1];
char RightZipAddOnFieldName[MAX_FIELD_NAME_LEN + 1];
char LeftCensusBlockFieldName[MAX_FIELD_NAME_LEN + 1];
char RightCensusBlockFieldName[MAX_FIELD_NAME_LEN + 1];
char LeftOddEvenFieldName[MAX_FIELD_NAME_LEN + 1];
char RightOddEvenFieldName[MAX_FIELD_NAME_LEN + 1];
char PlaceFieldName[MAX_FIELD_NAME_LEN + 1];
long iProcessFlag;
long iStartRow;
long iEndRow;
} UD_INFO, *pUD_INFO;
```

Chapter 3: OLE Automation API

Overview

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

In This Chapter . . .

Summary of Client Control Properties, Events, and Methods

The tables below define the properties, events and methods that come into play when developing a custom geocoding application based on the MapMarker Client Control and/or the OLE Automation API.

Input Property Name	Description	Included with Client Control Interface	Use without Client Control Interface
Firm	String: Firm name	✓	✓
Street	String: Street Address	✓	✓
City	String: City	✓	✓
State	String: State	✓	✓
Zip	String: ZIP Code	✓	✓
ZipPlus4	String: ZIP Code add-on	✓	✓
ServerName	String: Name of Server where MapMarker Server is running	✓	✓
ExactHouse	If TRUE, exact match on house number is required	✓	✓
ExactName	If TRUE, exact match on street name is required	✓	✓
ExactZIP	If TRUE, exact match on ZIP Code is required	✓	✓
ExpandSearch	If TRUE, MapMarker will expand the search area in which it looks for match candidates	✓	✓
ExpandSearchInState	If TRUE, the expanded search area is limited to the state	✓	✓
ExpandDistance	If ExpandSearch is set to TRUE, this specifies the radius of the search in miles from the finance area centroid.	✓	✓
MatchIntersections	If TRUE, street intersection matching is attempted.	✓	✓
LinearOffset	Double: defines the position of the geocoded point with respect to the street.	✓	✓

(Cont.)

Input Property Name	Description	Included with Client Control Interface	Use without Client Control Interface
PerpendicularSetback	Double: defines the position of the geocoded point with respect to corner.	✓	✓
Units	Integer: the units used in LinearOffset and PerpendicularSetback.	✓	✓
ShowPropertiesButton	If TRUE, the Show Properties button is visible (default).	✓	
ShowBorder	If TRUE, the border around the control's interface is visible (default).	✓	

Output Property Name	Description	Included with Client Control Interface	Use without Client Control Interface
Latitude	Double: the latitude value for a match candidate.	✓	✓
Longitude	Double: the longitude values for a match candidate.	✓	✓
Precision	Integer: a number that identifies match precision (street level, shape path, intersection or ZIP centroid).	✓	✓
numCandidates	Integer: the number of match candidates for the record.	✓	✓
StringBinding	String: RPC binding string used to connect to the server.	✓	✓
LastErrorCode	Long: last error code.	✓	✓
CensusBlockID	String: Census Block tabulation number.	✓	✓
ResultCode	String: geocoding result code	✓	✓

Event	Purpose	Included with Client Control Interface	Use without Client Control Interface
LatLongChanged()	This event is triggered when the Geocode button is pressed, the DoGeocode() method is called, or when a different candidate is selected in the Match Candidates list box. This event updates the latitude, longitude and precision properties.	✓	N/A
GeocodeEvent()	This event is triggered when the Geocode button is pressed and the whole geocoding process is completed. Note: The DoGeocode() method does not trigger this event.	✓	N/A

Method	Purpose	Included w/ Client Control Interface	Use without Client Control Interface
ClearDialogText()	Clears all visible text in the dialog, as well as these properties: firm, street, city, state, Zip, ZipPlus4, lastErrorCode, longitude, latitude, and precision.	✓	
RefreshDialog()	Tells the dialog to repaint. MapMarker Client Control will usually repaint automatically and you will not need to call it.	✓	
DoGeocode()	This method is the same as pressing the Geocode button. It implies that you must initialize these properties: firm, street, city, state, Zip, ZipPlus4, serverName.	✓	
GetCandidateAt()	Returns the line of text in the Match Candidate list box.	✓	
GetCandidateStreetAt()	Returns the street portion of the address as a string.	✓	

(Cont.) Method	Purpose	Included w/ Client Control Interface	Use without Client Control Interface
GetCandidateCityAt()	Returns the city portion of the address as a string.	✓	
GetCandidateStateAt()	Returns the state portion of the address as a string.	✓	
GetCandidateZIPAt()	Returns the ZIP portion of the address as a string.	✓	
GetCandidatePlus4At()	Returns the ZIP add on of the address as a string.	✓	
GetCandidatePrecisionAt()	Returns the precision for the match as a string. Precision refers to the quality of the match: to street level, shape path, intersection, or ZIP centroid.	✓	
GetCandidateLongitudeAt()	Returns the longitude as a double.	✓	
GetCandidateLatitudeAt()	Returns the latitude as a double.	✓	
GetCandidateCensusBlock IDAt()	Returns the Census Block ID as a string.	✓	
DoSetProperties()	Brings up a set of property pages for the Client Control.	✓	
SelectCandidateAt()	Highlights the specified candidate in the Match Candidate List box.	✓	
Connect()	Builds an RPC binding string to connect to the MapMarker Server.		✓
Disconnect()	Disconnects from the MapMarker Server.		✓
GeocodeAddress()	Attempts to geocode an address and build a list of candidates.		✓
GeocodeGetCandidates()	Gets all the candidate information from the GeocodeAddress call.		✓
GeocodeFreeSet()	Frees the server side information after the GeocodeAddress call.		✓
GeocodePostalCentroid()	Gets a ZIP Code or ZIP+4 centroid.		✓
GetLastErrorCode()	Returns the last error code (same as the LastErrorCode property).	✓	✓

(Cont.) Method	Purpose	Included w/ Client Control Interface	Use without Client Control Interface
GetStringBinding()	Returns a string that describes the connection to the MapMarker Server (same as StringBinding property).	✓	✓
GetName()	Returns the application name.	✓	✓
GetFullName()	Returns the full name and path of the application.	✓	✓

OLE Automation Methods

The following pages contain the OLE Automation method descriptions in alphabetical order. These descriptions have been updated from the MapMarker Product Guide to include usage examples.

ClearDialogText()

Purpose

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

Syntax

ClearDialogText()

Connect()

Purpose

This call builds an RPC binding string to connect to the MapMarker Server.

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

Syntax

Connect(NetworkAddress As String) As Boolean

Part	Description
Network Address	String: identifies the name or IP Address of the machine on which MapMarker Server is running. NULL or an empty string assumes MapMarker Server is running locally, and will attempt to use local RPC.

Returns

TRUE if successful, FALSE if not. The call to GetLastErrorCode can be used to determine what error occurred.

Disconnect()

Purpose

This call is used to disconnect from the MapMarker Server.

Syntax

Disconnect() As Boolean

Returns

TRUE if successful, FALSE if not. The call to GetLastErrorCode can be used to determine what error occurred.

DoGeocode()

Purpose

This is the same as clicking the Geocode button. Doing this assumes that the Firm, Street, City, State, Zip, ZipPlus4 and ServerName properties are initialized.

Syntax

DoGeocode()

DoSetProperties()

Purpose

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

Syntax

DoSetProperties()

GeocodeAddress()

Purpose

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

Syntax

```
GeocodeAddress(geocodeHandle As Long,
        Firm As String
        Street As String,
        City As String,
        State As String,
        ZIP As String
        Status As Integer
        NumCandidates As Integer
        NumCloseCandidates As Integer) As Boolean
```

Part	Description
geocodeHandle	Output: used to reference information about the geocoding operation in other calls.
Firm	Input: address component to be matched.
Street	Input: address component to be matched.
City	Input: address component to be matched.
State	Input: address component to be matched.
ZIP	Input: address component to be matched.
Status	Output: one of the following: (0) SINGLE_MATCH - a single close match was found for the input address. (1) MULTIPLE_MATCH - multiple match candidates were found and MapMarker could not choose between at least two of them. (2) NO_CLOSE_MATCH - candidates found, but none considered a close match. (3) NO_CANDIDATES - no candidates found for the input address. (4) SINGLE_INTERSECT_MATCH - an intersection match was found. (5) MULTIPLE_INTERSECT_MATCH - more than one intersection candidate was found. (6) NO_CLOSE_INTERSECT_MATCH - intersection candidates found, but none considered a close match. (7) NO_INTERSECT_CANDIDATES - no intersection candidates found. (8) POSSIBLE_INTERSECT_MATCH - possible close intersection match found.
NumCandidates	Output: total number of candidates found.
NumCloseCandidates	Output: number of close candidates.

Returns

TRUE if successful, FALSE if not. The call to GetLastErrorCode can be used to determine what error occurred. Some common error codes include:

Server not running error code

GEO_ENG_BAD_INPUT_ADDRESS where something in the input is missing

GEO_ENG_MISSING_DATABASE_ERRwhere MapMarker cannot find some things in the database path specified by the server. This could also mean you aren't licensed for street-level geocoding in that state (can try postal geocoding).

Usage Example

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

clearly. See the Programming Usage Example at the end of this chapter for a complete sample program that illustrates this function.

```
Private Sub cmdGeocode_Click()
Dim retVal As Boolean
Dim retCode
    retVal = objMM.GeocodeAddress(lngEngHandle, txtFirm, txtStreet,
txtCity, txtState, txtZip, status, numCandidates,
numCloseCandidates)
    If Not retVal Then
        'Error handling
   '... ...
    Else
   'Show geocoding status
        Select Case status
            Case 0
               lblStatus = "Single match"
            Case 1
                lblStatus = "Multiple match"
            Case 2
                lblStatus = "No close matches"
            Case 3
                lblStatus = "No candidates"
            Case 4
                lblStatus = "Single intersection match"
            Case 5
                lblStatus = "Multiple intersection match"
            Case 6
                lblStatus = "No close intersection matches"
            Case 7
                lblStatus = "No intersection candidates"
            Case 8
                lblStatus = "Possible intersections"
            Case Else
                lblStatus = Str$(status)
        End Select
    End If
End Sub
```

GeocodeFreeSet()

Purpose

This call frees all the server side information maintained after the GeocodeAddress call. The client should call this once they have finished with the address. Once the information is freed, the handle cannot be used again, unless the same value is returned by a later call to GeocodeAddress.

FreeSet releases the memory structure allocated to hold the candidate list. Whenever GeocodeAddress is invoked, a memory structure is allocated, the structure is populated with a list of possible candidates and a context handle to that structure is returned to the caller.

MapMarker can allocate a maximum of 1024 of these structures. To avoid running out of memory or blocking the MapMarker Server, call GeocodeFreeSet as soon as you have finished evaluating the candidate list.

Syntax

GeocodeFreeSet(geocodeHandle As Long) As Boolean

Part	Description
geocodeHandle	Output: used to reference information about the geocoding operation in other calls.

Returns

TRUE if successful, FALSE if not.

GeocodeGetCandidates()

Purpose

This call can be made to get all the needed candidate information from the GeocodeAddress call. This information will be stored in a collection class.

Syntax

GeocodeGetCandidates(geocodeHandle As Long,

NumCandidates As Integer) As Object

Part	Description
geocodeHandle	Output: used to reference information about the geocoding operation in other calls.
NumCandidates	Input: specifies the number of candidates that should be returned (i.e., the size of the collection).

Returns

A CAddressList collection of CAddress objects. The number of addresses to be stored in the collection is the input candidate number. This number may be less than or equal to the total number of candidates.

Usage Example

The following is a simplified version of a VB function for getting candidate addresses. UI handling, error handling and other details of coding are removed in order to show the processing flow more clearly. See the Programming Usage Example at the end of this chapter for a complete sample program that illustrates this function.

```
Private Sub cmdGetCand_Click()
Dim adrList As CAddressList
Dim adr As CAddress
Dim strAddr$, strCoords$, strPrec$, strCensId$
    Set adrList = objMM.GeocodeGetCandidates(lngEngHandle,
numCandidates)
    'Fill in the ListBox with the candidates
    lstCandidates.Clear
    For Each adr In adrList
        With adr
       'Set strAddr from .Street and other fields of adr
       'Set strCoords from .Latitude and .Longitude of adr
       'Set strPrec from .Precision of adr
       'Set strCensId from .CensusBlock of adr
        End With
        Set adr = Nothing
   'Add this candidate to the listbox
        lstCandidates.AddItem (strAddr & strCoords & strPrec &
strCensId)
    Next
    Set adrList = Nothing
End Sub
```

GeocodePostalCentroid()

Purpose

This call can be used to get a ZIP Code or ZIP+4 centroid.

Syntax

GeocodePostalCentroid(Zip As String,

Plus4 As String,

Longitude As Double,

Latitude As Double,

Precision As Integer

ResultCode As String) As Boolean

Part	Description
Zip	Input: ZIP Code address component
Plus4	Input: ZIP Code add-on address component
Longitude	Returns the longitude value.
Latitude	Returns the latitude value.
Precision	Returns one of the following: 30 (street-level match for an intersection address); 20 (street-level match for street address); 10 (Shape Path Centroid match); 3 (ZIP+4 Centroid match); 2 (ZIP+2 Centroid match); or 1 (ZIP Code Centroid match).
Result Code	Returns a value that is analogous to the GeoResult codes returned by MapMarker (defined in "Understanding Result Codes," in Chapter 6 in the MapMarker Product Guide.)

Returns

TRUE if successful, FALSE if not. A call to GetLastErrorCode can be used to determine what error occurred.

GetCandidateAt()

Purpose

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

Syntax

GetCandidateAt(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The string of tab-delimited text associated with the specified candidate.

GetCandidateCensusBlockIDAt()

Purpose

This call returns the Census Block ID of an address in the Match Candidates list box. The Census Block ID is a code of up to 15 digits and characters that describes the smallest of U.S. Census Bureau census units.

Syntax

GetCandidateCensusBlockIDAt(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The Census Block ID of the specified candidate.

GetCandidateCityAt()

Purpose

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

Syntax

GetCandidateCityAt(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The city of the specified candidate.

GetCandidateLatitudeAt()

Purpose

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

Syntax

GetCandidateLatitudeAt(Index As Integer) As Double

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The latitude of the specified candidate.

GetCandidateLongitudeAt()

Purpose

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

Syntax

GetCandidateLongitudeAt(Index As Integer) As Double

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The longitude of the specified candidate.

GetCandidatePlus4At()

Purpose

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

Syntax

GetCandidatePlus4At(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The ZIP Add on (Plus 4) of the specified candidate.

GetCandidatePrecisionAt()

Purpose

This call returns the precision portion of an address in the Match Candidates list box. The precision defines the type of match: street level, shape path, intersection, or ZIP centroid (either ZIP+4, ZIP+2 or ZIP Code).

Syntax

GetCandidatePrecisionAt(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The precision of the specified candidate.

GetCandidateResultCodeAt()

Purpose

This call returns the result code for an address in the Match Candidates list box. The code represents the type of match (single, multiple or ZIP centroid) and how precisely the GeoEngine matched to the address components (house number, street name, prefix, street type, city name, ZIP Code and whether it matched to the MapMarker Address Dictionary or user defined dictionary.)

Syntax

GetCandidateResultCodeAt(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The result code of the specified candidate.

GetCandidateStateAt()

Purpose

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

Syntax

GetCandidateStateAt(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The state of the specified candidate.

GetCandidateStreetAt()

Purpose

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

Syntax

GetCandidateStreetAt(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The firm followed by a tab and the street address of the specified candidate. Could also be the firm or street alone if there is no additional information.

GetCandidateZIPAt()

Purpose

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

Syntax

GetCandidateZIPAt(Index As Integer) As String

Part	Description
Index	The number from the zero-based index list that corresponds to the desired candidate.

Returns

The ZIP Code of the specified candidate.

GetFullName()

Purpose

This method returns the full name and path of the application.

Syntax

GetFullName()As String

GetLastErrorCode()

Purpose

This call returns the last error code. This is the same as the LastErrorCode property.

Syntax

GetLastErrorCode() As Long

Returns

The last error code. These are error codes from the geoerror.h file or from the RPC server.

GetName()

Purpose

This call returns the name of the application.

Syntax

GetName()As String

GetStringBinding()

Purpose

This call returns a string that describes the connection to the MapMarker Server. This is the same as the StringBinding property.

Syntax

GetStringBinding()As String

Returns

The client/server binding string.

RefreshDialog()

Purpose

This call redraws the MapMarker Geocoder Control's interface.

Syntax

RefreshDialog()

SelectCandidateAt()

Purpose

This function causes the MapMarker Client Control to highlight the specified candidate in the candidate list box.

Syntax

SelectCandidateAt()

OLE Automation Objects

CAddressList

The CAddress List collection is a pointer to a collection class that can be used to iterate through the candidate addresses. The CAddressList is a collection of CAddress objects that are returned by GeocodeGetCandidates(). Each CAddress object has the following properties:

Firm As String - The matched firm name

Street As String - The matched street address

City As String - The matched city

State As String - The matched state

Zip As String - The matched ZIP Code

plus4 As String - The matched ZIP Add on

Precision As Double - The precision of the matched coordinate

- 30 Street Level (Intersection)
- 20 Street Level
- 10 Shape Path Centroid
- 3 ZIP+4 Centroid
- 2 ZIP +2 Centroid
- 1 ZIP Code centroid

ResultCode As String - The matched result code

Longitude As Double - The longitude of the matched address

Latitude As Double - The latitude of the matched address

CensusBlock as String - Census Block ID of the matched candidate.

Programming Usage Example

The following is a sample user interface and corresponding Visual Basic code that illustrates the functions and usage of a geocoding control object.


```
Dim objMM As Object 'MapMarkr
Dim lngEngHandle&
Dim status As Integer
Dim numCandidates As Integer
Dim numCloseCandidates As Integer
Dim bConnected As Boolean
Dim bSettingModified As Boolean
Private Sub chkExtendedSearch_Click()
    bSettingModified = True
End Sub
Private Sub chkInState_Click()
    bSettingModified = True
End Sub
Private Sub chkMatchCentroid_Click()
    bSettingModified = True
End Sub
Private Sub chkMatchHouse_Click()
    bSettingModified = True
```

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

```
    ' Centroid geocoding does not create candidates
    cmdGetCand.Enabled = False
    Exit Sub
GeocodePostalCentroid_Error:
    MsgBox "Error Zip centroid geocoding"
End Sub
Private Sub cmdConnect_Click()
Dim retVal As Boolean
    If bConnected = False Then
        On Error GoTo Connect_Error
        retVal = objMM.Connect(txtServer)
        If retVal = True Then
            bConnected = True
            ' Only when connected, can do geocoding
            cmdGeocode.Enabled = True
            cmdCentroid.Enabled = True
            cmdDisconnect.Enabled = True
            cmdConnect.Enabled = False
        End If
    End If
    cmdRefresh_Click
    Exit Sub

Connect_Error:
    If txtServer = "" Then
        MsgBox ("Error connecting to local server")
    Else
        MsgBox ("Error connecting to server: " & txtServer)
    End If
End Sub
Private Sub cmdDisconnect_Click()
Dim retVal As Boolean
    If bConnected = True Then
        retVal = objMM.GeocodeFreeSet(lngEngHandle)
        retVal = objMM.Disconnect
        bConnected = False
        cmdGeocode.Enabled = False
        cmdCentroid.Enabled = False
        cmdGetCand.Enabled = False
        cmdConnect.Enabled = True
        cmdDisconnect.Enabled = False
        cmdFreeSet.Enabled = False
    End If
    cmdRefresh_Click
End Sub
Private Sub cmdFreeSet_Click()
```

```
Dim retVal As Boolean
    retVal = objMM.GeocodeFreeSet(lngEngHandle)
    'Candidates freed
    cmdGetCand.Enabled = False
    cmdFreeSet.Enabled = False
End Sub
Private Sub cmdGeocode_Click()
Dim retVal As Boolean
Dim retCode
Dim errorCode&
    ' User changed geocoding config parameters
    If bSettingModified = True Then
        retCode = MsgBox("Config parameters have been changed. Do
you want to use new parameters?", 3, "Street Geocoding")
        If retCode = vbYes Then
            cmdSet_Click
        ElseIf retCode = vbCancel Then
            Exit Sub
        End If
    End If
    lblLong = ""
    lblLat = ""
    lblPrec = ""
    lblStatus = ""
    lblCand = ""
    lstCandidates.Clear
    On Error GoTo GeocodeAddress_Error
    retVal = objMM.GeocodeAddress(lngEngHandle, txtFirm, txtStreet,
txtCity, txtState, txtZip, status, numCandidates,
numCloseCandidates)
    On Error GoTo 0
    If Not retVal Then
        errorCode = objMM.LastErrorCode
        If errorCode = 14 Then
            lblStatus = "Invalid address"
        'Sometimes .Connect returns true even server not
        'running, but .GeocodeAddress will return 1753
        ElseIf errorCode = 1753 Then
            If txtServer = "" Then
                lblStatus = "Local server is not running"
            Else
                lblStatus = "Server : " & txtServer & " is not
running"
            End If
        Else
```

```
            lblStatus = "Error geocoding, error code: " &
Str$(errorCode)
        End If
    Else
        Select Case status
            Case 0
               lblStatus = "Single match"
            Case 1
                lblStatus = "Multiple match"
            Case 2
                lblStatus = "No close matches"
            Case 3
                lblStatus = "No candidates"
            Case 4
                lblStatus = "Single intersection match"
            Case 5
                lblStatus = "Multiple intersection match"
            Case 6
                lblStatus = "No close intersection matches"
            Case 7
                lblStatus = "No intersection candidates"
            Case 8
                lblStatus = "Possible intersections"
            Case Else
                lblStatus = Str$(status)
        End Select
        If numCandidates > 0 Then
            lblCand = Str$(numCandidates)
            'Now .GetCandidates can be called to get
            'candidate information
            cmdGetCand.Enabled = True
            cmdFreeSet.Enabled = True
        Else
            cmdGetCand.Enabled = False
            cmdFreeSet.Enabled = False
        End If
    End If
    Exit Sub
GeocodeAddress_Error:
    lblStatus = "Error geocoding"
End Sub
Private Sub cmdGetCand_Click()
Dim adrList As CAddressList
Dim adr As CAddress
Dim strAddr$, strCoords$, strPrec$, strCensId$
    On Error GoTo GeocodeGetCandidates_Error
```

```
    Set adrList = objMM.GeocodeGetCandidates(lngEngHandle,
numCandidates)
    On Error GoTo 0
    'Fill in the ListBox with the candidates
    lstCandidates.Clear
    For Each adr In adrList
        With adr
            If IsEmpty(.Firm) Or .Firm = "" Then
                strAddr = .Street & ", " & .City & ", " & .State & "
" & .Zip & "-" & .plus4
            Else
                strAddr = .Firm & ", " & .Street & ", " & .City & ",
" & .State & " " & .Zip & "-" & .plus4
            End If
            strCoords = " (" & Format(.Latitude, "##0.0000") & ", "
& Format(.Longitude, "##0.0000") & ")"
            Select Case .Precision
                Case 30
                    strPrec = " Street-level (Xsect) "
                Case 20
                    strPrec = " Street-level "
                Case 10
                    strPrec = " Shape-path Cent. "
                Case 3
                    strPrec = " Zip+4 "
                Case 2
                    strPrec = " Zip+2 "
                Case 1
                    strPrec = " Zip centroid "
            End Select
            strCensId = .CensusBlock
        End With
        Set adr = Nothing
        lstCandidates.AddItem (strAddr & strCoords & strPrec &
strCensId)

    Next
    Set adrList = Nothing
    Exit Sub

GeocodeGetCandidates_Error:
    MsgBox "Error getting candidates"
End Sub
Private Sub cmdRefresh_Click()
    'Set UI from the properties of objMM
    With objMM
```

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

```
            End If
            If chkMatchStreet.Value = 1 Then
                .ExactName = True
            Else
                .ExactName = False
            End If
            If chkMatchCentroid.Value = 1 Then
                .ExactZip = True
            Else
                .ExactZip = False
            End If
            If chkExtendedSearch.Value = 1 Then
                .ExpandSearch = True
            Else
                .ExpandSearch = False
            End If
            If chkInState.Value = 1 Then
                .ExpandSearchInState = True
            Else
                .ExpandSearchInState = False
            End If
            If chkMatchXsec.Value = 1 Then
                .MatchIntersections = True
            Else
                .MatchIntersections = False
            End If
            .ExpandDistance = txtDistance
            .LinearOffset = txtOffsetLine
            .PerpendicularSetback = txtOffsetPerp
            bSettingModified = False
        End With
    End Sub
    Private Sub Form_Load()
        Set objMM = CreateObject("MAPMARKR.MapMarkrCtrl.1")
        bConnected = False
        bSettingModified = False
        cmdGeocode.Enabled = False
        cmdCentroid.Enabled = False
        cmdGetCand.Enabled = False
        cmdFreeSet.Enabled = False
        cmdDisconnect.Enabled = False
    End Sub
    Private Sub txtDistance_Change()
        bSettingModified = True
    End Sub
    Private Sub txtOffsetLine_Change()
```

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

Chapter 4: MapMarker Server RPC API

Overview

C programmers using Remote Procedure Call (RPC) now have a tool to develop custom geocoding applications that call the MapMarker Server, providing nationwide geocoding services for virtually any Windows desktop. The MapMarker RPC API is a collection of methods that allows you to connect to the MapMarker Server and geocode records from a client. This chapter describes the API and assists you in developing a custom geocoding application. We assume that you are familiar with RPC and how to connect to remote servers.

In This Chapter . . .

MapMarker Server RPC Interface

MapMarker Server runs as an NT service and can be stopped and started using the NT Service Control Manager. It also runs as a console application in Windows 95 or Windows NT. See the MM3xdoc.pdf.

The Server grabs a Local RPC connection and a TCP/IP socket with a dynamically assigned endpoint. It registers the endpoints in the machine's endpoint registry. A client can connect to the process locally using *ncalrpc* and remotely using a partially bound *ncacn_ip_tcp handle*.

The server uses RPC context_handles to store local information obtained during a call to GeocodeAddress. This enables the candidates and other information to be retrieved by the client after the geocoding is complete. The Geocoding Engine is not thread safe, and as a result, the server prevents multiple threads from using it by guarding the GEOCODEHANDLE with a mutex. The time involved in geocoding a single address is very small, so threads should never have to wait long.

Using the RPC API to Build a Remote Geocoding Application

The following files are included in the RPC toolkit:

- mm_dist.h (prototypes for RPC calls)
- mm_distc.c (client-side RPC stubs)
- mm_dist.idl (interface definition file).

These files are found in the geoeng\rpc subdirectory after installation.

When compiling your client application, you will need to include the first two listed files (mm_dist.h and mm_distc.c). Your application will also need to link against the Microsoft run time library for RPC (rpcrt4.lib).

API Function Calls

The remainder of this chapter describes each of the function calls in the RPC API.

Note: We assume that you have established an RPC connection to the MapMarker Server and are prepared to geocode.

GeocodeAddress()

Purpose

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

Syntax

```
long GeocodeAddress(GEOCODE_HANDLE *geocodeHandle,
                    char *pFirm,
                     char *pStreet,
                    char *pCity,
                    char *pState,
                     char *pZIP,
                    short relaxFlags,
                    double linearInset,
                    double perpendicularSetback,
                    short distUnit,
                    short relaxDistance,
                    short *pStatus,
                    short *pNumCandidates,
                    short *pNumCloseCandidates)
```

Parameters

geocodeHandle is a context handle needed to reference candidates and match status from a geocode address call.

Firm, Street, City, State, and ZIP are input address components to be matched.

RelaxFlags specifies values that control the match (i.e., in the Geocode dialog). These are used to override the server settings. Values are found in mmdist.h.

linearInset and perpendicularSetback are the same as in the System Preferences dialog.

RelaxDistance specifies the distance on a relaxed finance search.

DistUnit specifies the distance unit used for the linearOffset and perpendicularSetback. Possible values are found in geostd.h.

Status returns the result code for a match candidate.

NumCandidates is the number of match candidates for the address.

NumCloseCandidates is the number of close candidates for an address.

Returns

Zero if successful, otherwise one of the MapMarker error codes. Note: that following a successful geocode, you must free the candidates when done using the GeocodeFreeSet() call. Otherwise it will eventually stop the Server from accepting new requests to geocode.

If the call was successful, then the geocodeHandle is used to reference information about the geocoding operation in other calls. The user can specify MMD_DEFAULT_CFG for these to use the server values.

GeocodeGetCandidate()

Purpose

This call can be made to get all the needed candidate information from the GeocodeAddress call. If the status was SINGLE_MATCH, then this call with a candidateNumber of 0 would return the match and coordinate information for the matched address.

Using the IDL, the return values will need to be pre-allocated and have the following sizes:

char firm[MMD_FIRM_LEN]

char streetAddress[MMD_STREET_LEN]

char city[MMD_CITY_LEN];

char state[MMD_STATE_LEN];

char zip[MMD_ZIP_LEN];

char plus4[MMD_PLUS4_LEN];

char censusBlockID[MMD_CENSUS_BLOCK_LEN]

Syntax

```
long GeocodeGetCandidate(GEOCODE_HANDLE geocodeHandle,
        short candidateNumber,
        char *firm,
        char *streetAddress,
        char *city
        char *state,
        char *zip,
        char *plus4,
        double *longitude,
        double *latitude,
        short *precision,
        char *resultcode
        char *censusBlockID)
```

Parameters

geocodeHandle is a context handle needed to reference candidates and match status from a geocode address call.

Firm, streetAddress, city, state, zip, plus4, longitude, latitude, precision and resultcode are the output candidate address components.

Returns

This function will return zero on success, or one of the following values:

1 – The geocodeId was invalid or already freed.

2 – The candidateNumber is out of range.

GeocodeFreeSet()

Purpose

This call frees all the server side information maintained after the GeocodeAddress call. The client should call this once they have finished with the address. Once the information is freed, the handle cannot be used again, unless the same value is returned by a later call to GeocodeAddress.

FreeSet releases the memory structure allocated to hold the candidate list. Whenever GeocodeAddress is invoked, a memory structure is allocated, the structure is populated with a list of possible candidates and a context handle to that structure is returned to the caller.

MapMarker can allocate a maximum of 1024 of these structures. To avoid running out of memory or blocking the MapMarker Server, call GeocodeFreeSet as soon as you have finished evaluating the candidate list.

Syntax

short GeocodeFreeSet(GEOCODE_HANDLE*geocodeHandle)

Parameters

geocodeHandle is a context handle needed to reference candidates and match status from a geocode address call.

Returns

Zero if successful, and 1 if not.

GeocodePostalCentroid()

Purpose

This call can be used to get a ZIP Code or ZIP+4 centroid. The coordinates and precision are only changed if zero is returned.

Syntax

```
short GeocodePostalCentroid(char *zip,
        char *plus4,
        double *longitude,
        double *latitude,
        short *precision
        char *resultcode)
```

Parameters

Zip, plus4 are input address components.

Plus4, longitude, latitude, precision and result code are the returned results. pCoordPrecision of 0 means there was no match.

Returns

Zero if successful, and MapMarker error code if not.

GeocodeShutdown()

Purpose

Shuts down the MapMarker Server. Note: This will terminate all active sessions. The normal procedure is to shut down the MapMarker Server using the Windows NT Service Control Manager.

Syntax

```
void GeocodeShutdown()
```