Oracle® *inter*Media Locator

User’s Guide and Reference

Release 8.1.7

September 2000

Part No. A85334-01

Oracle *inter*Media Locator is a standards-based data management solution for delivery of Internet and mobile location-based services. Locator provides a complete *plug and play* infrastructure for wireless telecommunications, Internet Service Providers (ISPs), and Application Service Providers (ASPs).

Oracle *inter*Media Locator User’s Guide and Reference, Release 8.1.7

Part No. A85334-01

Primary Author: Rod Ward

Contributors: Frank Wang, Chuck Murray, Xavier Lopez, Susan Shepard, Brenda Silva

Contents

List of Figures

List of Tables

Send Us Your Comments

Oracle *inter*Media Locator User's Guide and Reference, Release 8.1.7

Part No. A85334-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: nedc_doc@us.oracle.com
- FAX: 603.897.3316 Attn: Oracle *inter*Media Locator Documentation
- Postal service:
  Oracle Corporation
  Oracle *inter*Media Locator Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This guide describes how to use Oracle *inter*Media Locator.

Oracle *inter*Media Locator requires Oracle8*i* or Oracle8*i* Enterprise Edition.

For information about the differences between Oracle8*i* and Oracle8*i* Enterprise Edition and the features and options that are available to you, see *Getting to Know Oracle8i*.

Audience

This guide is for anyone who is interested in storing, retrieving, and manipulating locator point data in an Oracle database, including developers of locator specialization services.

Organization

This guide contains the following chapters and appendixes:

Chapter 1	Introduces Oracle *inter*Media Locator; explains concepts.
Chapter 2	Describes the Oracle *inter*Media Locator functions, the geocoding service, and the locator operator, and provides examples of their use.
Chapter 3	Describes the generic geocoding interface to third-party geocoding software.
Appendix A	Describes how to run the sample application and includes the source program.
Appendix B	Lists exceptions raised and potential errors, their causes, and user actions to correct them.

Related Documents

Note: For information added after the release of this guide, refer to the online README.TXT file in your *ORACLE_HOME* directory. Depending on your operating system, this file may be in:

`ORACLE_HOME/md/doc/README.TXT`

Please see your operating-system specific installation guide for more information.

For the latest documentation, see the Oracle Technology Network Web site:

`http://technet.oracle.com/`

For more information about using this product in a development environment, see the following documents in the Release 8.1.7 Oracle8*i* documentation set:

- *Getting to Know Oracle8i*
- *Oracle8i Application Developer's Guide - Fundamentals*
- *Oracle8i Administrator's Guide*
- *Oracle8i Error Messages*
- *Oracle8i Utilities*
- *Oracle8i Concepts*
- *Oracle8i Tuning*
- *SQL*Plus User's Guide and Reference*

Conventions

In this guide, Oracle *inter*Media Locator is sometimes referred to as Locator.

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are also used in this guide:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
...	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface text	Boldface text indicates a term defined in the text.
italic text	Italic text is used for emphasis, for book titles, and variable names.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

Changes to This Guide

The following substantive changes were made to this guide since its previous version for release 8.1.6 on the Oracle Technology Network (OTN) Web site.

Other minor corrections and clarifications are also included.

Locator supports two new Oracle partners, MapQuest.com and whereonearth.com, who each provide geocoding services to complement your Oracle solution. See Chapter 1 for more information.

The generic geocoding interface to third-party geocoding software for batch geocoding is described. See Chapter 3 for more information.

New sample code is provided that describes how to use Locator to find the distance between two points. See Section A.2.2 for more information.

1 Introduction

Oracle *inter*Media Locator enables Oracle8*i* to support online Internet-based geocoding facilities and batch geocoding services for locator applications and proximity queries.

Geocoding represents addresses and locations of interest (postal codes, demographic regions, and so forth) as geometric factors (points). These enable distances to be calculated and sites to be represented graphically in Web, data warehousing, customer information system, and enterprise resource planning applications. Geocoding services can be used to add the exact location (latitude and longitude) of points of interest to existing data files stored in Oracle8*i*.

Oracle *inter*Media Locator supports the leading online and batch geocoding services including MapXtreme from MapInfo Corporation, Centrus from Qualitative Marketing Software, MapQuest destination information solutions from MapQuest.com ("MapQuest"), and GeoZip from whereonearth.com Ltd.

MapInfo Corporation, Qualitative Marketing Software, MapQuest.com, and whereonearth.com currently provide the online and batch geocoding services for the Locator features. Each service offers a number of free geocoding calls at its Web site for trial purposes for online geocoding, and geocoding service software for batch geocoding. Locator users need to consent to the vendor policies and possibly register with them:

MapInfo Corporation: `http://www.MapMarker.com/`

Qualitative Marketing Software: `http://www.centrus-software.com/oracle/`

MapQuest.com: `http://www.mapquest.com/`

whereonearth.com: `http://www.whereonearth.com/`

During registration for online geocoding services, you are asked to create your own user ID and password. Please make a note of them for embedding into your sample geocoding service because the user ID / password combination is required for each geocoding call. Your free account is limited to a small number of address records per day.

Should you require the ability to geocode larger data sets, or for further information, contact:

- MapInfo technologies to complement your Oracle solution; call 1.800.FAST-MAP (1.800.327.8627); or send e-mail to custserv@mapinfo.com (see their Web site for more specific geographic contact information)
- QMSoft technologies to complement your Oracle solution; call QMSoft at 1.800.782.7988; or send e-mail to oracle@qmsoft.com
- MapQuest.com technologies to complement your Oracle solution; call 1.888.MAPQUEST (1.888.627.7837) or in Europe, (31) 70.426.2660; or send e-mail to info@mapquest.com
- whereonearth.com technologies to complement your Oracle solution; call +44 (0) 207 246 1400; or send e-mail to enquiries@whereonearth.com

These companies' Web sites will also have detailed documentation about the vendor-specific parameter information of the Locator features, such as match code or error code. Because Oracle provides an interface to facilitate the geocoding functions, you should contact the vendors for any questions regarding the geocoding services.

See the Oracle *inter*Media Locator Release Notes (*ORACLE_HOME* / md / doc / README.txt) for additional information about the geocoding services provided by these Oracle partners.

Oracle *inter*Media Locator also supports server-based geocoding and data scrubbing operations for data warehouse applications.

Using simple location queries, Oracle *inter*Media Locator lets Web and other applications retrieve information based on distance. For example, using a set of geocoded address data and simple query-by-text or query-by-map operations, users can use a Web browser-based application, enter a distance, and identify the nearest location from a specific address or reference point on a map. For example, Oracle *inter*Media Locator applications can help you locate stores, offices, distribution points, and other points of interest based on their distance from a given postal (zip) code, address, or other reference point.

Oracle *inter*Media Locator supports geocoding, storage, and retrieval of geocoded, spatial-point data in Oracle8*i* databases. Oracle *inter*Media Locator is not designed to be an end-user application. Its features consist of:

- A Locator object type that describes and supports only the point-geometry object type
- A geocode result object type that describes the geocode result definition
- A call interface described by two geocode result functions used for geocoding spatial data that also contains the output geocode result object and the Locator geometry object
- A function to better estimate the index level for use with the spatial locator index for within-distance queries that use a radius distance greater than 100 miles
- A procedure to create a spatial locator index on the column where the spatial information is stored in the geocoded table that is used by the locator operator
- A locator operator that uses geometric intersection algorithms and the spatial locator index for performing within-distance queries

Based on this implementation, this Oracle *inter*Media Locator release supports:

- Geocoding spatial-point data by providing the means to add a geocoded address column or objects to existing tables and storing the geocoded address column or objects locally in the Oracle8*i* universal database server
- Inserting and retrieving geocoded address data
- Performing simple within-distance text- or map-based queries on the geocoded data

Some example uses of these Locator features are the following:

- Locate stores, offices, or distribution points based on their distances from a given reference point such as an address or postal code.
- Locate restaurants or hotels within a given point-to-point distance using a person's specific address, or current location on a map, such as at a tourist information center.

These features enable database designers to extend existing application databases with geocoded, spatial-point data, or to build new geocoded spatial-point applications. Web application developers can build specialized Web-enabled Locator applications.

For additional information, see the following:

- Chapter 2 describes the Web-based Locator functions, the geocoding service, and the Locator operator along with basic examples of using the Locator object types.

- Chapter 3 describes the generic interface to third-party geocoding software that lets users geocode an entire address table through third-party batch geocoding services.
- Appendix A describes a number of sample scripts that are installed and that you can modify and run.
- Appendix B describes Oracle *inter*Media Locator exceptions and error messages.

2 Locator Functions

2.1 Locator Implementation

The implementation of Oracle *inter*Media Locator functions consists of a set of object types, an index method type, and an operator on these types. A geometry is stored in a single row in a column of type SDO_GEOMETRY. Spatial index creation and maintenance is done using data definition language (DDL) (CREATE, ALTER, DROP) and data manipulation language (DML) (INSERT, UPDATE, DELETE) statements.

2.1.1 Locator Structures

The geometric description of a Locator object is stored in a single row in a column of type SDO_GEOMETRY. This row is in a user-defined table that has one primary key column (or a set columns that constitute a primary key) and optionally one or more attribute columns.

The object type SDO_GEOMETRY is defined as:

```
Create Type SDO_GEOMETRY as object (
SDO_GTYPE NUMBER,
SDO_SRID NUMBER,
SDO_POINT SDO_POINT_TYPE,
SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,
SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
```

The attributes have the following semantics:

- SDO_GTYPE - The type of the geometry. The valid geometry type is:

  `1 = POINT`
  The geometry type must always be 1.

- SDO_SRID - Spatial reference identifier. This is always NULL.
- SDO_POINT - An object type with attributes X, Y, and Z; all of type NUMBER represented as longitude, latitude, and NULL, respectively.
- SDO_ELEM_INFO - Always NULL.
- SDO_ORDINATES - Always NULL.

2.2 Results Definition and Geocode Functions

This section contains a description of the geocode result object type definition and the call interface described by two geocode functions as shown in Table 2–1.

Table 2–1 Locator Functions and Procedures

Type/Function	Description
GEOCODE_RESULT object	Geocode result object definition
GEOCODE1 function	Geocode function that contains a lastline field; but no city, state, or postal code (zip) fields
GEOCODE1 function	Geocode function that contains city, state, and postal code (zip) fields, but no lastline field

GEOCODE_RESULT Object

Format

```
create type GEOCODE_RESULT AS OBJECT(
  matchcode varchar2(16),
  firmname    varchar2(512),
  addrline      varchar2(512),
  addrline2    varchar2(512),
  city            varchar2(512),
  state          varchar2(512),
  zip             varchar2(5),
  zip4           varchar2(4),
  lastline       varchar2(512),
  county        varchar2(32),
  block          varchar2(32),
  loccode       varchar2(16),
  cart            varchar2(16),
  dpbc          varchar2(16),
  lotcode       varchar2(16),
  lotnum        varchar2(16)
);
/
```

Description

Describes the geocode result definition.

Parameters

matchcode	Match result, indicating the quality of a match
firmname	Firm name
addrline	Address line 1
addrline2	Address line 2
city	City
state	State
zip	Postal (zip) code
zip4	Plus 4 digit zip code
lastline	City, state, zip code
county	Federal information processing standards (FIPS) county code
block	Census block identifier
loccode	Location code
cart	Carrier route (postal service)
dpbc	Delivery point bar code
lotcode	Line of travel code
lotnum	Line of travel number

Usage Notes

In their implementation of Locator, geocode vendors may make use of all or most fields in the GEOCODE_RESULT table. See the vendor's documentation for a complete description of this object and the fields used.

Exceptions

Application-specific exceptions:

http_error, -20000

geocoder_error, -20001

unit_error, -20003

GEOCODE1 Function (with lastline field)

Format

```
function GEOCODE1(url       in varchar2,
                  proxy     in varchar2,
                  name      in varchar2,
                  pwd       in varchar2,
                  firmname  in varchar2,
                  addrline  in varchar2,
                  addrline2 in varchar2,
                  lastline  in varchar2,
                  mm        in varchar2,
                  stdaddr   out MDSYS.GEOCODE_RESULT,
                  location  out MDSYS.SDO_GEOMETRY) return varchar2;
pragma restrict_references(GEOCODE1, WNDS, WNPS);
```

Description

Used for geocoding and includes a lastline field that contains city, state, and zip code information.

Parameters

url	Vendor Web site for geocoding: for example, `www.centrus-software.com/oracle/geoservice.dll`
proxy	Security protection mechanisms (firewall) address, NULL or ('') if none
name	Customer name (for accounting)
pwd	Password (for accounting)
firmname	Firm name
addrline	Address line 1
addrline2	Address line 2
lastline	Contains city, state, postal (zip) code, and zip4 information
mm	Matchmode; a string telling the vendor which match mode to use, such as STANDARD, NORMAL, and so forth See vendor sites for more information.
stdaddr	Standard address object or output geocode result object (defined previously)
location	Locator geometry object, SDO_GEOMETRY, containing latitude and longitude information

Return Value

This return value is the error code returned as a string by the geocode vendor; typically, the string contains an error code and a message, such as 0:SUCCESS. See the specific vendor documentation for more information.

Usage Notes

The lastline field contains the city, state, and postal (zip) code information.

Exceptions

None.

Examples

Example 1: Geocode a single record interactively.

```
-- Geocode a single record interactively.
set serveroutput on
set timing on
set pagesize 50000

declare
  geo_result MDSYS.GEOCODE_RESULT;
  geom MDSYS.SDO_GEOMETRY;
  result varchar2(255);
begin
  result := geocoder_http.GEOCODE1(
              'http://www.centrus-software.com/oracle/geoservice.dll',
              'www-proxy.us.acme.com',
              'user', 'password',
              'oracle','1 oracle dr','', 'nashua NH 03062',
              'tight',
              geo_result, geom);
  dbms_output.put_line(result);
exception
when geocoder_http.http_error then
   dbms_output.put_line('Internet problem - cannot connect');
when geocoder_http.geocoder_error then
   dbms_output.put_line('Geocoder problem - contact vendor');
when others then
   dbms_output.put_line('Oracle Error - check your PL/SQL');
end;
/
```

Example 2: Geocode a table in batch mode using the entire object.

```
-- See how to create this sample table using the file nh_cs.sql.
-- Geocode a table in batch mode using the entire object.

-- HOW TO CUSTOMIZE IT FOR YOUR USE:
-- 1. Change the select statement in the declaration section to match
--    your input table.
--    If you are placing the geocode result into the same table, make sure
--     rowid is selected; if you are geocoding into a different table, make sure
--    the primary keys are selected.
--
-- 2. In the update call at the end, if you are placing all your results
--    back in the same table, use update ... where rowid = r.rowid;
--    otherwise, use insert into ... where pk = r.pk;
--
-- 3. Exception handling:
```

```
--    The routine generates HTTP_ERROR and GEOCODER_ERROR.
--    HTTP_ERROR corresponds to a transmission problem.
--    GEOCODER_ERROR is when an address record cannot be matched by the
--     geocoder from the vendor Web site, and the result you get back is likely
--    to be null.
--    You should decide how to handle these errors according to your
--    own needs.
--    The GEOCODER_ERROR exception can be examined in the result variable.
--
declare
  CURSOR crs is
     select company, address, city, state, zipcode, rowid from
nh_computer_stores;
  standard_address MDSYS.GEOCODE_RESULT;
  geom_location MDSYS.SDO_GEOMETRY;
  result varchar2(255);
begin
  for r in crs loop
   begin
    result := geocoder_http.GEOCODE1(
      'http://www.centrus-software.com/oracle/geoservice.dll',
      'www-proxy.us.acme.com',
      'user','password',
      r.company,
      r.address, '',
      r.city, r.state, r.zipcode,
      'normal',
      standard_address,
      geom_location);
   exception
   when geocoder_http.geocoder_error then
      dbms_output.put_line('Geocoder error, continuing');
   when others then
      dbms_output.put_line('HTTP or server error, quit');
      exit;
   end;
   update nh_computer_stores
      set std_addr = standard_address, location = geom_location
      where rowid = r.rowid;
<<end_loop>>
    null;
  end loop;
end;
/
```

Example 3: Geocode a table in batch mode using fields in the object.

```
-- Geocode a table in batch mode using fields in the object.

-- HOW TO CUSTOMIZE IT FOR YOUR USE:
-- 1. Change the select statement in declaration section to match
--    your input table.
--    If you are placing the geocode result into the same table, make sure
--     rowid is selected; if you are geocoding into a different table, make sure
--    the primary keys are selected.
--
-- 2. In the update call at the end, if you are placing all your results
--    back in the same table, use update ... where rowid = r.rowid;
--    otherwise, use insert into ... where pk = r.pk;
--
-- 3. Exception handling:
--    The routine generates HTTP_ERROR and GEOCODER_ERROR.
--    HTTP_ERROR corresponds to transmission problem.
--    GEOCODER_ERROR is when an address record cannot be matched by the
--     geocoder from the vendor Web site, and the result you get back is likely
--    to be null.
--    You should decide how to handle these errors according to your
--    own needs.
--    The GEOCODER_ERROR exception can be examined in the result variable.
--
declare
  CURSOR crs is
     select company, address, city, state, zipcode, rowid from
nh_computer_stores;
  standard_address MDSYS.GEOCODE_RESULT;
  geom_location MDSYS.SDO_GEOMETRY;
  result varchar2(255);
begin
  for r in crs loop
   begin
    result := geocoder_http.GEOCODE1(
      'http://www.centrus-software.com/oracle/geoservice.dll',
      'www-proxy.us.acme.com',
      'user','password',
      r.company,
      r.address, '',
      r.city, r.state, r.zipcode,
      'normal',
      standard_address,
      geom_location);
   exception
   when geocoder_http.geocoder_error then
      dbms_output.put_line('Geocoder error, continuing');
   when others then
      dbms_output.put_line('HTTP or server error, quit');
      exit;
   end;
   update nh_computer_stores
      set std_street = standard_address.address,
      std_city = standard_address.city,
      std_state = standard_address.state,
      std_zip = standard_address.zip,
      std_zip4 = standard_address.zip4,
      location = geom_location
      where rowid = r.rowid;
<<end_loop>>
    null;
  end loop;
end;
/
```

GEOCODE1 Function (with city, state, and postal code (zip) fields)

Format

```
function GEOCODE1(url        in varchar2,
                  proxy      in varchar2,
                  name       in varchar2,
                  pwd        in varchar2,
                  firmname   in varchar2,
                  addrline   in varchar2,
                  addrline2  in varchar2,
                  city       in varchar2,
                  state      in varchar2,
                  zip        in varchar2,
                  mm         in varchar2,
                  stdaddr    out MDSYS.GEOCODE_RESULT,
                  location   out MDSYS.SDO_GEOMETRY) return varchar2;
pragma restrict_references(GEOCODE1, WNDS, WNPS);
```

Description

Used for geocoding and includes city, state, and postal (zip) code fields.

Parameters

url	Vendor Web site for geocoding: for example, `www.centrus-software.com/oracle/geoservice.dll`
proxy	Security protection mechanisms (firewall) address, NULL or ('') if none
name	Customer name (for accounting)
pwd	Password (for accounting)
firmname	Firm name
addrline	Address line 1
addrline2	Address line 2
city	City name
state	State name
zip	Postal (zip) code
mm	Matchmode; a string telling the vendor which match mode to use, such as STANDARD, NORMAL, and so forth See vendor sites for more information.
stdaddr	Standard address object or output geocode result object (defined previously)
location	Locator geometry object, SDO_GEOMETRY, containing latitude and longitude information

Return Value

The return value is the error code returned as a string by the geocode vendor; typically, the string contains an error code and a message, such as 0:SUCCESS. See the specific vendor documentation for more information.

Usage Notes

The city, state, and postal (zip) fields replace the lastline field described in the previous function.

Exceptions

None.

Examples

See the examples in the previous GEOCODE1 function description.

2.3 Estimate Level and Spatial Locator Index

This section describes the ESTIMATE_LEVEL function and the spatial locator index. If you must use the ESTIMATE_LEVEL function, call this function prior to creating the spatial locator index. The spatial locator index must be created before you can use the locator operator described in Section 2.4.

Table 2–2 Locator ESTIMATE_LEVEL Function and Spatial Locator Index

Function/Procedure	Description
ESTIMATE_LEVEL	Estimates an appropriate index_level parameter value when most of your LOCATOR_WITHIN_DISTANCE queries use a radius distance value that exceeds 100 miles.
SETUP_LOCATOR_INDEX	Creates the spatial locator index.

ESTIMATE_LEVEL

Format

```
function ESTIMATE_LEVEL(radius1 in number,
                        radius2 in number) return integer;
```

Description

Calculates an index_level parameter value for use in the SETUP_LOCATOR_INDEX procedure.

> **Note:** Only call this function if most of your LOCATOR_WITHIN_DISTANCE queries use a radius distance value greater than 100 miles; otherwise, the default value of 13 is appropriate as the index_level parameter value.

Parameters

radius1	Small radius in miles.
radius2	Large radius in miles.

Return Value

The return value is the appropriate index_level parameter value to use in the SETUP_LOCATOR_INDEX procedure.

Usage Notes

If you expect to use a large radius distance for queries that is greater than 100 miles, you should call this function to determine the most appropriate index_level parameter value for your data.

A LOCATOR_WITHIN_DISTANCE query with a circular radius distance greater than 100 miles actually degenerates into an ellipse with two semiaxes (radii). Therefore, this function has two parameters, radius1 to represent the small semiaxis and radius2 to represent the large semiaxis of the ellipse. For Oracle8*i* release 8.1.7, you should provide the same value for both radius1and radius2 parameters.

If you must call this function, call this function after you geocode your data and before you create your spatial locator index. A more appropriate index_level parameter value is expected to give you better performance on your data.

Exceptions

Application-specific exceptions:

unit_error, -20004

Examples

Create a setup spatial locator index.

```
select geocoder_http.estimate_level(200,200) from dual;
9
```

SETUP_LOCATOR_INDEX

Format

```
procedure SETUP_LOCATOR_INDEX(tabname    in varchar2,
                              colname    in varchar2,
                              index_level in number := 13);
```

Description

Creates the spatial locator index.

Parameters

tabname	Table name where the spatial information is stored
colname	Column name where the spatial information is stored within 'tabname'
index_level	Value determined by calling the ESTIMATE_LEVEL function when the radius distance exceeds 100 miles and a better index level is required to improve performance on your data The default value is 13.

Return Value

None.

Usage Notes

This procedure creates a metadata table called USER_SDO_GEOM_METADATA under the invoker's or current user's schema. It creates a special domain index of type spatial_index. The name of the index is:

```
substr((tabname,1,5)||'_'substr(colname,1,5)||'_idx'||_HL6N1$
```

Do not delete these extra tables after creating the index.

This procedure must be executed to create the spatial locator index for the geocoded table before you can use the LOCATOR_WITHIN_DISTANCE operator; otherwise, an error message is returned indicating no spatial locator index is created. For example:

```
ERROR at line 1:
ORA-20000: Interface Not Supported without a Spatial Index
ORA-06512: at "MDSYS.SDO_3GL", line 184
ORA-06512: at line 1
```

Usually, you do not need to modify the value of the index_level parameter if most of your LOCATOR_WITHIN_DISTANCE queries are using a radius distance value of 100 miles or less. However, to achieve better performance on your data, you can change this value depending on the most popular radius distance for most of your LOCATOR_WITHIN_DISTANCE queries. To estimate a better value for the index_level parameter, call the ESTIMATE_LEVEL function. In this case, you must call the ESTIMATE_LEVEL function before you create your spatial locator index.

Exceptions

None.

Examples

Create a setup spatial locator index.

```
procedure SETUP_LOCATOR_INDEX('cust_table', 'location', 13);
```

2.4 Locator Operator

This section describes the function used when working with the Locator object type.

Table 2–3 Locator Operator

Function	Description
LOCATOR_WITHIN_DISTANCE	Determines if two points are within a specified geometric distance from one another.

LOCATOR_WITHIN_DISTANCE

Format

LOCATOR_WITHIN_DISTANCE(T.Column MDSYS.SDO_GEOMETRY, aGeom MDSYS.SDO_GEOMETRY, params VARCHAR2);

Description

Uses geometric intersection algorithms and a spatial index to identify the set of spatial points that are within some specified geometric distance (radius distance) of a given point of interest (center of a circle).

Parameters

params	Determines the behavior of the operator
	Valid keywords and their semantics are described as follows:
distance	Required; the radius distance value
units	Required; the unit value; can be mile, ft (feet), or meter

Return Value

The expression LOCATOR_WITHIN_DISTANCE(arg1, arg2, arg3) = ‘TRUE’ evaluates to TRUE for point pairs that are within the specified distance apart, and FALSE otherwise.

Usage Notes

- The distance around a point of interest describes a circle and this distance is defined as the minimum radius distance between these two points.
- The operator must always be used in a WHERE clause and the condition that includes the operator should be an expression of the form:

```
LOCATOR_WITHIN_DISTANCE(arg1, arg2,
‘distance = <some_dist_val>, units=mile’) = ‘TRUE’.
```

- It is required that T.Column have a spatial locator index built on it. See Section 2.3 for more information.
- LOCATOR_WITHIN_DISTANCE( ) is not supported for spatial joins.
- The default unit is latitude and longitude. Therefore, you should always specify a unit such as: mile, ft, or meter.

Exceptions

None.

Examples

Example 1: Perform a simple point query.

```
SELECT A.GID FROM POINTS A WHERE LOCATOR_WITHIN_DISTANCE
(A.Geometry, :aGeom, ‘distance = 10 units=mile’) = ‘TRUE’ ;
```

Example 2: Perform a computer store query.

```
Rem
Rem $Header: geolocate.sql 14-sep-98.11:51:16 pfwang Exp $
Rem
Rem geolocate.sql
Rem
Rem  Copyright (c) Oracle Corporation 1998. All Rights Reserved.
Rem

-- This routine dynamically creates a geometry of interest,
-- for example, an Oracle office location. Then, it queries against the
-- NH_COMPUTER_STORES table to find out how many computer stores are
-- within a certain distance (radius) of the office. In this case, the
-- distance is 10 miles.

set serveroutput on
set pagesize 50000

declare
  standard_address MDSYS.GEOCODE_RESULT;
  geom_location    MDSYS.SDO_GEOMETRY;
  result           varchar2(255);
  type cur_type is ref cursor ;
  crs cur_type;
begin
  result := geocoder_http.geocode1(
              'http://www.centrus-software.com/oracle/geoservice.dll',
              'www-proxy.us.acme.com',
              'user', 'password',
              'Oracle','1 Oracle Drive','', '03062',
              'tight', standard_address, geom_location);
  if (instr(upper(result),'SUCCESS') = 0) then
     raise geocoder_http.geocoder_error;
  end if;
  open crs for
    'select company from nh_computer_stores where '||
       'MDSYS.LOCATOR_WITHIN_DISTANCE(location, :1, ''distance=10
units=Mile'')=''TRUE'''
  using geom_location;
  loop
    fetch crs into result;
    exit when crs%NOTFOUND;
    dbms_output.put_line(result);
  end loop;
  close crs;
exception
when geocoder_http.http_error then
   dbms_output.put_line('Internet problem - cannot connect');
when geocoder_http.geocoder_error then
   dbms_output.put_line('Geocoder problem - contact vendor');
when others then
   dbms_output.put_line('Oracle Error - check your PL/SQL');
end;
/
```

3 Generic Geocoding Interface

This chapter describes a generic interface to third-party geocoding software that lets users geocode their address information stored in database tables and obtain standardized addresses and corresponding location information as instances of predefined object types. This interface is part of the geocoding framework in the Oracle Spatial and Oracle *inter*Media Locator products.

A geocoding service is used for converting tables of address data into standardized address, location, and possibly other data. Given a geocoded address, one can then perform proximity or location queries using a spatial engine, such as Oracle Spatial or Oracle *inter*Media Locator, or demographic analysis using tools and data from Oracle's business partners.

Once data has been geocoded, users can perform location queries on this data. In addition, geocoded data can be used with other spatial data such as block group, postal code, and county code for association with demographic information. It is now possible for decision support, customer relationship management, supply chain analysis, and other applications to use spatial analyses as part of their information gathering and processing functions. Results of analyses or queries can be presented as maps, in addition to tabular formats, using third-party software integrated with Oracle *inter*Media Locator.

This chapter describes a set of interfaces and metadata schema that enables geocoding of an entire address table, or a single row. It also describes the procedures for inserting or updating standardized address and spatial point data into another table (or the same table). The third-party geocoding service is assumed to have been installed on a local network and to be accessible through standard communication protocols such as sockets or HTTP.

3.1 Locator Implementation: Benefits and Limitations

Oracle *inter*Media Locator contains a set of application programming interface (API) functions that allows the integration of Oracle Spatial with third-party geocoding products and Web-based geocoding services. A database user can issue a standard SQL call or construct PL/SQL routines to geocode an address, and retrieve the spatial and standardized address objects, both of which are defined as Oracle database object types. Users have the option of storing these in the database, or using the spatial objects in Locator functions for Euclidean within-distance queries.

The APIs offer great flexibility in extracting information from existing relational databases. Data conversion procedures are minimal. A geocode result also returns an additional set of information; there is no requirement to use all the information, and the application can decide which fields to extract and where to store them. However, to use the full range of features of Oracle Spatial or Oracle *inter*Media Locator, it is recommended that the Spatial object be stored as returned.

The existing Locator service is Web-based and requests are formatted in HTTP. Thus, each request in SQL must contain the URL of the Web site, proxy for the firewall (if any), and user account information on the service provider's Web site. An HTTP approach potentially limits the utility or practicality of the service when dealing with large tables or undertaking frequent updates to the base address information. In such situations, use a batch geocoding service made available within an intranet or local area network. The following sections describe the interface for a facility that can include the existing HTTP-based solution.

3.2 Generic Geocoding Client

A fast, scalable, highly available, and secure Java Virtual Machine (Java VM, or JVM) is integrated in the Oracle8*i* database server. The Java VM provides an ideal platform on which to deploy enterprise applications written in Java as Java Stored Procedures (JSPs), Enterprise Java Beans (EJBs), or Java Methods of Oracle8*i* object types.

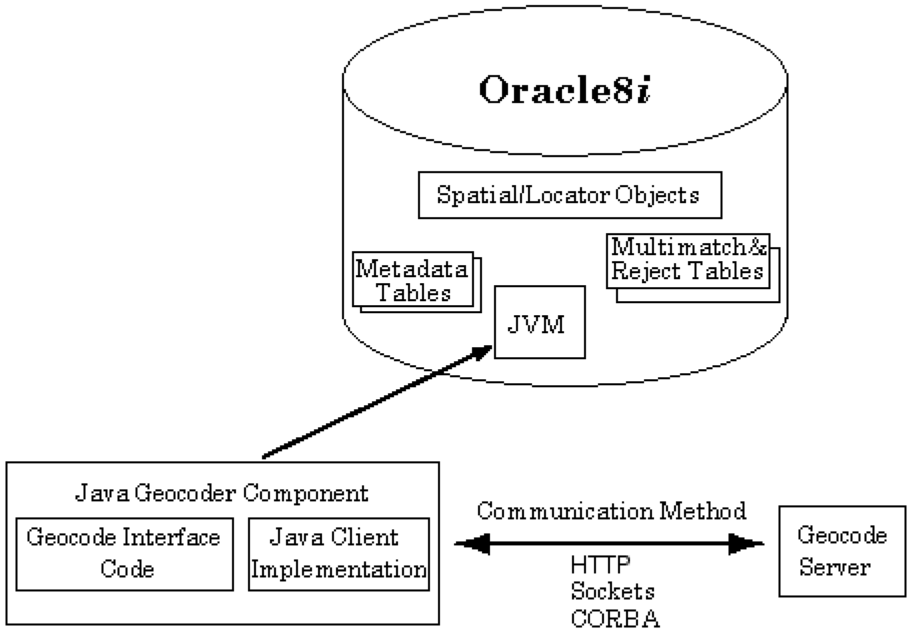
Therefore, any client geocoder component written in Java can be embedded in the Oracle8*i* database as a JSP. This JSP interface can perform either one-record-at-a-time or batch geocoding. Java stored procedures are published using PL/SQL interfaces; thus, the generic geocoding interface can be compatible with existing Locator APIs.

The stored procedures have an interface, oracle.spatial.geocoder, that must be implemented by each vendor whose geocoder is integrated with Oracle Spatial and Oracle *inter*Media Locator. The procedures also require certain object types to be defined and metadata tables to be populated. The object types, metadata schema, and the geocoder interface are described in further detail in the following sections.

Although the database user MDSYS oversees all data types, operators, and schema objects for Oracle Spatial and Oracle *inter*Media Locator, the geocoding metadata must exist in each user's schema. Each user of the geocoder service must have tables that implement the metadata schema.

Figure 3–1 shows the Oracle geocoding framework.

Figure 3–1 Oracle Geocoding Framework



3.3 Geocoder Metadata

The metadata describes the properties of the geocoding server, the location and structure of the address data to be geocoded, and the nature and storage location of the geocoding results. Other relevant information may include the name of the server machine, the port to connect, and so on. Together, these constitute the initialization parameters and are stored in metadata tables under the user's own schema.

At client initialization, a data dictionary lookup is performed to locate the necessary metadata.

Batch geocoding lets the user simultaneously geocode many records from one table. Batch geocoding requires the following:

- Geocoding server setup, instructing the client where and how to connect to the geocoding service.
- Associating input fields and output fields with columns in the database tables. This is called the schema setup.
- Specifying how to handle geocoding situations such as rejects, multiple matches, or exceptions.

Thus, the metadata table consists of a task ID, geocoding information, and schema information. The task ID is a primary key that identifies the initialization parameters for a particular geocoding task. For example, geocoding a table of customers is one task, while geocoding a table of customer inquiries is a separate task.

The metadata is stored in a table named GEOCODE_TASK_METADATA, which is defined as follows:

```
Create table geocode_task_metadata (
  task_id  NUMBER,  -- primary key
  geocoder_info  MYSYS.GEOCODE_SERVER_PROPERTY_TYPE,
  schema_info  MDSYS.GEOCODE_SCHEMA_PROPERTY_TYPE
);
```

Note the following about the GEOCODE_TASK_METADATA table:

- The metadata is divided into a server object (described in Section 3.3.1) and a schema object (described in Section 3.3.2).
- Each object is identified by a unique *task_id* value.

3.3.1 Server Properties

The *geocoder_info* property column of the GEOCODE_TASK_METADATA table contains information describing the characteristics of the server, including machines, ports, and vendor-specific information.

The GEOCODE_SERVER_PROPERTY_TYPE type is defined as follows:

```
create type geocode_value_array as
  varray(1024) of varchar2(64)
/
create type geocode_server_property_type as object
```

```
(
  servers  geocode_value_array,
  protocol  varchar2(32),
  property_name  geocode_value_array,
  property_value  geocode_value_array,
  reject_level  integer,
  batch_size  integer
)
/
```

Note the following about the GEOCODE_SERVER_PROPERTY_TYPE definition:

- *servers* is an array of character strings each in the form *Machine:Port* that uniquely identifies the geocoding service on the network. This also supports multiple services on the same network by providing an array of servers. Some geocoders, for example, can switch to secondary servers in the case of failures.
- *protocol* allows different transport mechanisms, such as HTTP or socket.
- Additional *property_name* and *property_value* arrays allow customization for unique geocoder processing options. They are not intended to be used for name or password information, because a local geocoding service usually does not require this information.
- *reject_level* is a vendor-specific value that defines the criteria for rejecting a record. It is up to the implementation of the Java interface to interpret the value.
- *batch_size* indicates how many records to send to the geocoder at one time.

3.3.2 Geocoding Input and Output Specification

The *schema_info* property column of the GEOCODE_TASK_METADATA table specifies the set of columns that makes up an address in the table to be geocoded, the table and columns into which the geocoded results are stored, and where rejected record data and multiple matches are stored.

The GEOCODE_SCHEMA_PROPERTY_TYPE type uses columns of type GEOCODE_TABLE_COLUMN_TYPE to describe the address fields in the input (table to be geocoded) and output (table containing geocoded results). The two types are defined as follows:

```
create type geocode_table_column_type as object
(
  firm  varchar2(32),
  street  varchar2(32),
  street2  varchar2(32),
  cty_subdivision  varchar2(32),
  city  varchar2(2332),
  country_subdivision  varchar2(32),  --state
  country  varchar2(32),
  postal_code  varchar2(32),
  postal_addon_code  varchar2(32),
  lastline  varchar2(32),
  col_name  geocode_value_array,
  col_value  geocode_value_array
)
/

create type geocode_schema_property_type as object
(
  language  varchar2(32),
  character_set  varchar2(32),
  in_table  varchar2(32),
  in_table_cols  geocode_table_column_type,
  out_table  varchar2(32),
  out_table_cols  geocode_table_column_type,
  out_sdo_geom  varchar2(32),
  out_geo_result  varchar2(32),
  in_primary_key  varchar2(32),
  out_foreign_key  varchar2(32),
  DML_option  varchar2(16),
  multi_match_table  varchar2(32),
  reject_table  varchar2(32),
  batch_commit  varchar2(5)
)
/
```

Note the following about the GEOCODE_TABLE_COLUMN_TYPE and GEOCODE_SCHEMA_PROPERTY_TYPE definitions:

- *language* and *character_set* are for internationalization.
- *in_table* identifies the name of the input address table (for example, 'CUSTOMERS').
- *in_table_cols* identifies the standard set of fields for geocoding. The fields in the object are standard, and *lastline* is redundant with the combination of *city*, *state*, *postal_code*, and *postal_addon_code*. Only one (*lastline*, or the combination of *city*, *state*, *postal_code*, and *postal_addon_code*) should be specified.
- *out_table* and *out_table_cols* have the same meaning as *in_table* and *in_table_cols*, except that these are the column names where the results are stored. Either a

subset or all of the *out_table_cols* fields can be null. *out_table_cols* and *geocode_result* contain similar information, that is, the standardized (corrected) address in case of successful geocoding. Users can choose to store the standardized address in two forms, expanded into a set of columns or as a single object.

- If the actual address definition differs from the fields in the GEOCODE_TABLE_COLUMN_TYPE definition, adjust the field mappings and insert null values as needed. For example, assume an input table *customers* defined as follows:

  ```
   (custname  varchar2(32),
    company  varchar2(32),
    street  varchar2(64),
    city  varchar2(32),
    state  varchar2(32),
    country  varchar2(32),
    zip  varchar2(9))
  ```

  In the GEOCODE_SCHEMA_PROPERTY_TYPE column definition, the *in_table_cols* attribute value would be specified as: *geocode_table_column_type('company', 'street', null, null, 'city', 'state', 'country', 'zip', null, null, null, null).*

  The *col_name* and *col_value* information will be used for feature enhancement for individual geocoding services.

- *out_sdo_geom* and *out_geo_result*: SDO_GEOMETRY and GEOCODE_RESULT are the two database objects for storing a standard set of geocoded results, including standardized address and latitude/longitude information. If you are using Oracle Spatial, it is required that SDO_GEOMETRY objects be stored in the database. MDSYS.GEOCODE_RESULT exists in the current Locator implementation and is defined as follows:

  ```
  Create type geocode_result as object  (
    matchcode  varchar2(16),
    firmname  varchar2(512),
    addrline  varchar2(512),
    addrline2  varchar2(512),
    city  varchar2(512),
    state  varchar2(512),
    zip  varchar2(5),
    zip4  varchar2(4),
    lastline  varchar2(512),
    county  varchar2(32),
    block  varchar2(32),
  ```

```
  loccode  varchar2(16),
  cart  varchar2(16),
  dpbc  varchar2(16),
  lotcode  vrchar2(16),
  lotnum  varchar2(16)
);
```

- *in_primary_key* and *out_foreign_key*: Using a primary key and foreign key pair is a way to associate the input records to the output records, and is essential when the database stores the output results. Even if the input table and output table are the same, a primary key and foreign key pair (essentially the same column: for example, 'ID' or 'ROWID') must be specified. There is no restriction on the data type, because no manipulation of the data is needed.
- *DML_option* specifies whether to insert geocoded data into a new row in the result table ('INSERT') or update existing rows in the table ('UPDATE'). If *in_table* is the same as *out_table*, then *DML_option* must be 'UPDATE', because adding new rows in an existing table is unnecessary. If *in_table* is different from *out_table* and if 'UPDATE' is specified, *out_table* must have partial records available for primary and foreign key lookup. This permits the service to locate the exact row to update with the new objects.
- *multi_match_table* and *reject_table* are table names where the primary key of the multiple matches and rejected records are stored. If these tables do not exist, they will be created automatically. Automatic creation is the preferred approach due to the fixed structure. *reject_table* will be created with a primary key column type in the input table, a match code column, and an optional error message column. *multi_match_table* will contain a primary key, SDO_GEOMETRY, and GEO_RESULT. If these fields are null, no table will be created and no multiple matches will be returned.
- *batch_commit* is a string containing 'TRUE' or 'FALSE', indicating if a commit operation should be performed after each batch. If 'FALSE' is specified, a large rollback segment will be needed for large address table geocoding.

3.3.2.1 Multiple Matches and Rejected Records

Tables can be specified to store multiple matches (*multi_match_table*) and rejected records (*reject_table*) during batch geocoding. The primary key will be a user-specified field from the original table. Hence, any single column can be used. Currently, no composite primary keys are supported.

If a single address results in multiple matches, after the batch processing, you can examine *multi_match_table* and select the correct entries for the original data rows. For example, you can create a table in the following format:

```
create table <user-defined multimatch table> (
  pk  <same data type as in input table>,
  location  mdsys.sdo_geometry,
  std_addr  mdsys.geocode_result
);
```

The match code in the geocode result object indicates the failure during geocoding. The rejection level is used in determining if a record has failed the geocoding. If a record has failed and *reject_table* is defined, the primary key (specified by the user) is inserted into a rejection table. The interpretation of rejection level is left to the programmer. *reject_table* can be defined in the following format:

```
create table <user-defined reject table> (
  pk  <same data type as in input table>,
  matchcode  varchar2(64),
  errcode  varchar2(128)
);
```

3.4 Metadata Helper Class

The geocoder metadata is comprehensive. To accelerate development and deployment, Oracle offers a sample class, oracle.spatial.geocoder.Metadata, to allow easy access (read and write) to these objects. Also, SELECT and INSERT SQL statements are constructed automatically for the caller. See the class implementation code for details.

3.5 Single-Record and Interactive Geocoding

Geocoding a row in a table is required when updating or inserting data in the address table. One way to maintain consistency between the base address table and the table of geocoded results is to use a trigger to call the geocoding function. The Java interface method geocode1( ) will take the primary key to perform the geocoding task and insert or update the geocoded information into the specified table.

The GEOCODER_HTTP package functions are still supported for single-record geocoding. In addition, you are able to pass an address in as a parameter, and get back an array of matches. The Java interface takes a metadata structure (see the GEOCODE_SCHEMA_PROPERTY_TYPE definition in Section 3.3.2) and an address structure, and returns an array of this same address structure:

```
create type geocode_record_type as object
(
  firm  varchar2(40),
  street  varchar2(40),
  street2  varchar2(40),
  city_subdivision  varchar2(40),
  city  varchar2(40),
  country_subdivision  varchar2(40),
  country  varchar2(40),
  postal_code  varchar2(40),
  postal_addon_code  varchar2(40),
  lastline  varchar2(80),
  latitude  number,
  longitude  number
);
```

After performing geocoding, it will return an array (SQL collection type) of such structures as possible matches. In this method, no database table or schema is accessed. This method can enable interactive applications such as store locators.

3.6 Java Geocoder Service Interface

Each geocoder independent software vendor (ISV) must implement the following geocoder interface in order to integrate their products with Oracle Spatial and Oracle *inter*Media Locator.

The interface is defined as follows:

```
// Geocoder Interface
package oracle.spatial.geocoder;

public interface GeocoderInterface {
  public void geocode(int taskId)
    throws oracle.spatial.geocoder.GeocoderException, java.sql.SQLException;
  public void geocode1(int taskId, BigDecimal pkVal)
   throws oracle.spatial.geocoder.GeocoderException, java.sql.SQLException;

// … other geocode1 functions with different pkVal types

  public ARRAY interactive_geocode(STRUCT meta, STRUCT inAddr)
    throws oracle.spatial.geocoder.GeocoderException, java.sql.SQLException;
}

// Geocoder Exception Class
package oracle.spatial.geocoder;
```

```
public class GeocoderException extends java.lang.Exception {
  public GeocoderException() {}
  public GeocoderException(String mesg)
  {
    super(mesg);
  }
}
```

Further details, including some of the actual implementation, will be provided to developers by Oracle.

3.7 Enabling Third-Party Geocoders

For customers to implement an Oracle solution with any vendor's Java client, they will have to download a copy of the Java client from the geocoder vendor's Web site, link the geocoder interface package with the vendor's code, and then upload the resulting JSP into the Oracle JVM. Once enabled, the Java client resides on the vendor's server and can provide the required services.

To load a client into the database, invoke the Oracle8*i* loadjava utility, and the Java geocoding method will be exposed as a SQL function call.

The vendor-specific geocoder interface implementation can be owned by any schema, such as MDSYS, a DBA account, or an account determined by the customer or vendor. The owner must grant the appropriate EXECUTE privileges to PUBLIC or some set of users of the service.

A
Sample Programs

Oracle *inter*Media Locator includes a number of scripts that you can modify and run.

A.1 Sample Scripts

Sample Oracle *inter*Media Locator scripts are available in the following directory after you install this product:

```
$ORACLE_HOME/md/demo/geocoder
```

These scripts consist of the following files:

- geohttp.sql

  This file contains two parts. One part is for running a geocode function in interactive mode and the other is for running the geocode function in batch mode.

  – Interactive mode.

    See Example 1 in “GEOCODE1 Function (with lastline field)” on page 2-6 for a listing of this part of the file.

  – Batch mode.

    You must update the setup tables in the nh_cs.sql file before you run geohttp.sql in batch mode. See Example 2 in “GEOCODE1 Function (with lastline field)” on page 2-7 or Example 3 in “GEOCODE1 Function (with lastline field)” on page 2-9 for a listing of this part of the file.

- geoindex.sql

  This file contains:

  - A function named ESTIMATE_LEVEL to better estimate the index level for use with the spatial locator index for within-distance queries that use a radius distance greater than 100 miles. See the example in "ESTIMATE_LEVEL" on page 2-15 for a listing of this file.
  - A procedure statement named SETUP_LOCATOR_INDEX. This statement builds a setup spatial locator index on the location column that contains the spatial information within the cust_table table where the spatial information is stored. See the example in "SETUP_LOCATOR_INDEX" on page 2-17 for a listing of this file.

- geolocate.sql

  This file contains a routine that dynamically creates a geometry of interest and then queries against the NH_COMPUTER_STORES table to find out how many stores are within a 10-mile radius of the office. See Example 2 in "LOCATOR_WITHIN_DISTANCE" on page 2-19 for a listing of this file.

A.2 Sample Code

Oracle *inter*Media Locator includes sample code that you can modify and run.

A.2.1 Using Oracle *inter*Media Locator Services to Find Nearest Neighbor

Oracle *inter*Media Locator services support the within_distance operator. To facilitate a nearest-neighbor search, you must write a PL/SQL routine to find a number of geometries, and subsequently sort them by distance. The shortest distance between the point of interest and a neighboring point gives the nearest neighbor.

The following code sample illustrates how to find the top three points using Oracle *inter*Media Locator services. Assume the Porsche dealer and spatial tables are as follows:

```
create table porsche_dealer (
id number constraint pk_id primary key,
dealer_name varchar2(50),
address varchar2(50),
lastline varchar2(50),
phone varchar2(12),
fax varchar2(12));

create table porsche_spatial (
id constraint fk_id references porsche_dealer(id),
geo_address mdsys.geocode_result,
location mdsys.sdo_geometry,
map ordsys.ordimage,
last_modified date);
```

These two tables contain the database on US Porsche dealers. The Porsche dealer table contains dealer attribute data. The Porsche spatial table contains the geocoded location and result for each record in the Porsche dealer table. The goal is to find the top three Porsche dealers around a given customer location. This customer location (in address form) must first be geocoded. After the initial geocoding, the location object (sdo_geometry) is populated with the location for the center of the search. The following sample code illustrates how to find the top three points, using the Oracle *inter*Media Locator features:

```
procedure find_top3
found3 boolean;
dist number;
type cursor_type is ref cursor;
addr mdsys.geocode_result;
loc mdsys.sdo_geometry;
crs cursor_type;
dist number;
cnt number;
i number;
tmp number;

id_a dbms_sql.number_table;
company_a dbms_sql.varchar2_table;
address_a dbms_sql.varchar2_table;
lastline_a dbms_sql.varchar2_table;
phone_a dbms_sql.varchar2_table;
fax_a dbms_sql.varchar2_table;
x_a dbms_sql.number_table;
y_a dbms_sql.number_table;
dist_a dbms_sql.number_table;
no1 number;
no2 number;
no3 number;

begin

-- populate the location object by dynamically geocoding an address
-- geocode_http.geocode1(..., loc, addr);
-- ...

found3 := false;
-- keep increasing distance until you find 3 points
```

```
dist := 0.25;
while found3 = false loop

begin
dist := dist + 0.5;
open crs for ’select a.id, b.geo_address.firmname, b.geo_address.addrline,
b.geo_address.lastline, a.phone, a.fax, ’||
’b.location.sdo_point.x, b.location.sdo_point.y from ’||
’porsche_dealer a, porsche_spatial b where ’ ||
’a.id=b.id and ’||
’mdsys.locator_within_distance(b.location,’||
’:1,’’distance=’|| dist || ’’’)=’’TRUE’’’
using loc;
cnt := 1;
loop
fetch crs into id_a(cnt), company_a(cnt), address_a(cnt), lastline_a(cnt),
phone_a(cnt), fax_a(cnt), x_a(cnt), y_a(cnt);
exit when crs%NOTFOUND;
dbms_output.put_line(company_a(cnt)||’/’||address_a(cnt)||’/’||lastline_a(cnt));
cnt := cnt + 1;
end loop;
close crs;

-- dbms_output.put_line(cnt);
if cnt >= 4 then
found3 := true;
end if;
exception when NO_DATA_FOUND then

-- htp.print(’Start all over again’);
-- dbms_output.put_line(’new radius=’||dist);
if dist > 100 then
exit;
end if;
end;
end loop;
cnt := cnt - 1;

-- find the top 3 candidates
no1 := 1;
no2 := 2;
no3 := 3;
for i in 1 .. cnt loop
```

```
-- dbms_output.put_line(x_a(i) ||’,’|| y_a(i));
-- calculate distance

dist_a(i) := (loc.sdo_point.x - x_a(i))*(loc.sdo_point.x - x_a(i)) +
(loc.sdo_point.y - y_a(i))*(loc.sdo_point.y - y_a(i));

end loop;
for i in 4 .. cnt+1 loop

-- order the 3 numbers in the bucket
if dist_a(no1) > dist_a(no2) then
tmp := no2;
no2 := no1;
no1 := tmp;
end if;

if dist_a(no1) > dist_a(no3) then
tmp := no1;
no1 := no3;
no3 := tmp;
end if;

if dist_a(no2) > dist_a(no3) then
tmp := no2;
no2 := no3;
no3 := tmp;
end if;

if (i > cnt) then
exit;
end if;
if dist_a(i) < dist_a(no3) then
no3 := i;
end if;
end loop;

dbms_output.put_line(company_a(no1)||’\’||address_a(no1)||’\’||lastline_a(no1));
dbms_output.put_line(company_a(no2)||’\’||address_a(no2)||’\’||lastline_a(no2));
dbms_output.put_line(company_a(no3)||’\’||address_a(no3)||’\’||lastline_a(no3));
end;
```

A.2.2 Using Oracle *inter*Media Locator Services to Find Distance Between Two Points

This function is provided as a sample function with an open source. It calculates the true earth distance between two latitude/longitude points in meters. It assumes the earth to be a true spherical object, and the distance returned represents the distance on a great circle on the sphere. Results will have a small percent of inaccuracy (0.5% or less) depending on the point locations on earth.

Input: two SDO_GEOMETRY objects, each with SDO_POINT field filled, including x and y values.

The return value is the distance on earth in meters.

No exception is thrown, because this is a sample program illustrating the algorithm of the code.

```
function earth_distance(a mdsys.sdo_geometry, b mdsys.sdo_geometry) return
number as
  R number;
  c number;
  s1 number;
  s2 number;
  t1 number;
  t2 number;
  l number;
  ac number;
begin
  R := 6371.2;
  c := 3.14159265359 / 180.0;
  s1 := a.sdo_point.x;
  t1 := a.sdo_point.y;
  s2 := b.sdo_point.x;
  t2 := b.sdo_point.y;

  s1 := s1 * c;
  t1 := t1 * c;
  s2 := s2 * c;
  t2 := t2 * c;
  ac := cos(t1)*cos(t2)*cos(s2-s1)+sin(t1)*sin(t2);
  l := R * acos(ac) * 1000.0;
  return l;
end earth_distance;
```

B

Exceptions and Error Messages

B.1 Exceptions

This appendix describes the geocode HTTP package exceptions.

B.1.1 Geocode HTTP Package Exceptions

The following exceptions are associated with the geocode HTTP package.

http_error EXCEPTION
PRAGMA EXCEPTION_INIT(http_error, -20000)

Cause: This exception is raised when an HTTP transmission error occurs.

Action: The HTTP server may be down or the communications link may be down. Try again several times until successful, or try again later.

geocoder_error EXCEPTION
PRAGMA EXCEPTION_INIT(geocoder_error, -20001)

Cause: This exception is raised when a geocode vendor error occurs. This error is raised when a row cannot be matched by the geocode vendor and the result returned is likely to be null.

Action: Check with the specific vendor returning this exception to help diagnose the underlying problem and determine a solution.

unit_error EXCEPTION
PRAGMA EXCEPTION_INIT(unit_error, -20003)

Cause: This exception is raised when a unit conversion error occurs.

Action: A unit value is not recognized. Check your unit value for compliance.

radius_error EXCEPTION
PRAGMA EXCEPTION_INIT(unit_error, -20004)

Cause: This exception is raised when a negative radius value is used.

Action: Change the radius value to a positive value.

Index