Oracle Spatial

User Conference

April 29, 2010

Hyatt Regency Phoenix

Phoenix, Arizona USA
Developing Location-Enabled Applications: Oracle Spatial Geocoding and Routing Engines
Geocoding

• The geocoding process
• Geocoding data model
• Geocoding functions
• The SDO_GEO_ADDR structure
• Examples
• Structured Address Geocoding
• Reverse geocoding
What is Geocoding?

- Geocoding takes a textual address as input and returns a longitude/latitude coordinate associated with the address.
- Geocoding can have various levels of precision:
  - House number
  - Street
  - Postal code
  - Town
- Geocoding is used in many application areas:
  - Business Finders
  - Routing and directions
  - Mapping
The Geocoding Process

Geocoder

Address Parsing

Searching and Cleansing

Coordinate Generation

Reference Data for Geocoding

Address Structure

Street and place names

Geometries

Address to Match

Coordinates + Corrected Address
Computing House Location

- Range Based Geocoding
  - Streets have ranges of addresses
  - Location for a given address is interpolated based on the start and end values of the range
• Point Address Based Geocoding
  • Streets have ranges of addresses
  • Each address may also have the exact long/lat stored
  • Location for a given address is found with an exact match or interpolated based on the start and end values of the range
Oracle Spatial Geocoding

- Two APIs are provided
  - SQL: using PL/SQL functions
  - XML: sending XML requests to a web service
- Users must purchase data from Oracle Partners to use the Spatial geocoding functions
  - Data available from NAVTEQ
  - Other providers: format the data to meet the data model of the Geocoder
- Sample data is provided at Oracle partner sites. Links are provided from Spatial OTN web site:
Geocoding Tables

**Metadata tables**

- Describe the organization of the geocoding data
- Define address structure and parsing rules

**Data tables**

- Can have multiple sets of data
- One per country, or multiple countries per set
- Typically one set per country, with a country suffix

**Data country XX**

- GC_COUNTRY_PROFILE
- GC_PARSER_PROFILE
- GC_PARSER_PROFILEAFS
- GC_ROAD_xx
- GC_ROAD_SEGMENT_xx
- GC_AREA_xx
- GC_POSTAL_CODE_xx
- GC_POI_xx
- GC_INTERSECTION_xx
Geocoding Metadata

- **GC_COUNTRY_PROFILE**
  - Describes the organizational structure of a country
  - Administrative levels, Languages
  - Suffix of data tables for the country

- **GC_PARSER_PROFILEAFS**
  - Describes the address structure in each country
  - XML notation

- **GC_PARSER_PROFILES**
  - Defines keywords and common abbreviations
  - Can also store common spelling mistakes
Country Profile

- **GC_COUNTRY_PROFILE** table stores country profile information
  - Administrative area hierarchy definition
    - Number of admin levels
    - Optional admin levels
  - National languages for the country
  - Table-name suffix used by the data tables
Address Format Structure

- GC_PARSER_PROFILEAFS table stores the XML definition of postal-address formats
  - An XML string describes each address format for a specific country

```xml
<address_line>
  <street_address>
    <house_number> …</house_number>
    <street_name>
      <prefix /> <base_name /> <suffix /> <street_type />
      <special_format> …</special_format>
      …</street_name>
  </street_address>
</address_line>
```
Parser Profiles

• **GC_PARSER_PROFILES**
  • Defines keywords and common abbreviations
  • Can also store common spelling mistakes
  • `SDO_KEYWORDARRAY('US', 'USA', 'UNITED STATES', 'UNITED STATES OF AMERICA', 'U.S.A.', 'U.S.')`
    • OUTPUT_KEYWORD: US
  • `SDO_KEYWORDARRAY('AV', 'AVENUE', 'AVE', 'AVEN', 'AVENU', 'AVN', 'AVNUE', 'AV.','AVE.')`
    • OUTPUT_KEYWORD: AV
  • `SDO_KEYWORDARRAY('40TH', 'FORTIETH')`
    • OUTPUT_KEYWORD: 40TH
Geocoding Data

- **GC ROAD xx**
  - Street names and house numbers. Used for name searches

- **GC ROAD SEGMENT xx**
  - Street segments with geometries and house numbers. Used for computing coordinates

- **GC AREA xx**
  - Administrative areas

- **GC POSTAL CODE xx**
  - Postal codes

- **GC POI xx**
  - Points of interest

- **GC INTERSECTION xx**
  - US only: intersection based-searches.
Geocoder Functions and Data

- Address to Match
- Coordinates + Corrected Address
- SDO_GCDR package
- Geocoder Metadata
- Geocoder Data
The geocoding functions provided are:

- **SDO_GCDR.GEOCODE**
  - Returns an SDO_GEO_ADDR object with address and geocoding process information

- **SDO_GCDR.GEOCODE_ALL**
  - Returns an array of SDO_GEO_ADDR objects with address and geocoding process information

- **SDO_GCDR.GEOCODE_AS_GEOMETRY**
  - Returns an SDO_GEOMETRY object with the location of the geocoded address
Street Name Structure

- Street Name has 4 basic parts
  - North Access Road East
- Base Name
  - Access
- Type
  - Road (Rd), Street (St), Avenue (Av), etc.
- Prefix
  - North (N), South (S), East (E), West (W)
- Suffix
  - North (N), South (S), East (E), West (W)
Match mode is how close the specified address matches the stored address

- **EXACT** – All attributes in address match storage
- **RELAX_STREET_TYPE** – Allows match if street type (Street, Road, Blvd, Ave) doesn’t match
- **RELAX_POI_NAME** – Allows match if point-of-interest name doesn’t match (Central Park = Central Pk)
- **RELAX_HOUSE_NUMBER** – House number and street type can be different
Match mode

• RELAX_BASE_NAME - The base name of the street, the house number, and the street type can be different from the data used for geocoding. For example, Crosse Street can match with Cross Road

• RELAX_POSTAL_CODE, DEFAULT - The postal code, base street name, house number, and street type can be different from the stored data

• RELAX_BUILTUP_AREA, RELAX_ALL – Semantics of RELAX_POSTAL_CODE, but also relaxes city or town name as long as match is in the same county
Resulting address:  
SDO_GEO_ADDR

- Structure returned by functions GEOCODE() and GEOCODE_ALL()
- Contains the latitude and longitude of the address, but also:
  - Corrected and completed address details
  - Indications on the precision of the match
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONGITUDE</td>
<td>Geocoded longitude</td>
</tr>
<tr>
<td>LATITUDE</td>
<td>Geocoded latitude</td>
</tr>
<tr>
<td>MATCHCODE</td>
<td>Numeric match information</td>
</tr>
<tr>
<td>ERRORMESSAGE</td>
<td>How address elements matched</td>
</tr>
<tr>
<td>MATCHVECTOR</td>
<td>How address elements matched</td>
</tr>
<tr>
<td>TABLE COLUMNS</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STREET_NAME</td>
<td>The full name of the street</td>
</tr>
<tr>
<td>SETTLEMENT</td>
<td>The city or town</td>
</tr>
<tr>
<td>REGION</td>
<td>The state or province</td>
</tr>
<tr>
<td>POSTALCODE</td>
<td>Normal postal code</td>
</tr>
<tr>
<td>POSTALADDONCODE</td>
<td>Postal code extension</td>
</tr>
<tr>
<td>FULLPOSTALCODE</td>
<td>Full postal code</td>
</tr>
<tr>
<td>HOUSENUMBER</td>
<td>House number: 123 in 123 Main St</td>
</tr>
<tr>
<td>BASENAME</td>
<td>Street base name: <em>Main</em> in 123 Main Street</td>
</tr>
<tr>
<td>STREETTYPE</td>
<td>St for Street, Ave for Avenue, Rd for Road, etc</td>
</tr>
<tr>
<td>STREETPREFIX</td>
<td>Prefix: S in 123 S Main Street</td>
</tr>
<tr>
<td>STREETSUFFIX</td>
<td>Suffix: NW in 123 Main Street NW</td>
</tr>
<tr>
<td>SIDE</td>
<td>Side of street</td>
</tr>
<tr>
<td>PERCENT</td>
<td>Percentage value representing how far along the street you are when going</td>
</tr>
<tr>
<td></td>
<td>from lower to higher street addresses</td>
</tr>
<tr>
<td>EDGEID</td>
<td>Optional edge information if topological</td>
</tr>
</tbody>
</table>
## Matchcode: how did we match the input

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Exact match: the city name, postal code, street name, street type (and suffix or prefix or both, if applicable), and house number match</td>
</tr>
<tr>
<td>2</td>
<td>The city name, postal code, street name, and house number match, <strong>but the street type, suffix, or prefix does not match</strong></td>
</tr>
<tr>
<td>3</td>
<td>The city name, postal code, and street base name match, <strong>but the house number does not match</strong></td>
</tr>
<tr>
<td>4</td>
<td>The city name and postal code match, <strong>but the street name does not match</strong></td>
</tr>
<tr>
<td>10</td>
<td>The city name matches, <strong>but the postal code does not match</strong></td>
</tr>
<tr>
<td>11</td>
<td>The postal code matches the data used for geocoding, <strong>but the city name does not match</strong></td>
</tr>
</tbody>
</table>
Endormessage:
How precise is the match

- String of 17 characters.
- Each character corresponds to one address element.
- Each character indicates whether we matched on this address element.
- ‘?’ indicates no match on that element.
- Some positions not used
<table>
<thead>
<tr>
<th>Position</th>
<th>Meaning</th>
<th>Value When Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>House or building number</td>
<td>#</td>
</tr>
<tr>
<td>6</td>
<td>Street prefix</td>
<td>E</td>
</tr>
<tr>
<td>7</td>
<td>Street base name</td>
<td>N</td>
</tr>
<tr>
<td>8</td>
<td>Street suffix</td>
<td>U</td>
</tr>
<tr>
<td>9</td>
<td>Street type</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>Secondary unit</td>
<td>S</td>
</tr>
<tr>
<td>11</td>
<td>Built-up area or city</td>
<td>B</td>
</tr>
<tr>
<td>14</td>
<td>Region</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>Country</td>
<td>C</td>
</tr>
<tr>
<td>16</td>
<td>Postal code</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>Postal add-on code</td>
<td>A</td>
</tr>
</tbody>
</table>
Matchvector

- Also a string of 17 characters.
- Each position corresponds to one address element – just like ERRORMESSAGE
- Each position can have 4 values

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MATCHED</td>
<td>Address element specified and matched</td>
</tr>
<tr>
<td>1</td>
<td>ABSENT</td>
<td>Element not specified and not replaced</td>
</tr>
<tr>
<td>2</td>
<td>CORRECTED</td>
<td>Element specified but not matched, and was replaced with the correct value</td>
</tr>
<tr>
<td>3</td>
<td>IGNORED</td>
<td>Element specified, not matched and not replaced</td>
</tr>
<tr>
<td>4</td>
<td>SUPPLIED</td>
<td>Element missing, filled from the database.</td>
</tr>
</tbody>
</table>
Demonstration Title Here
Example: Street Level Match

```sql
SELECT SDO_GCDR.GEOCODE('SPATIAL',
  SDO_KEYWORDARRAY('Clay Street', 'San Francisco, CA'),
  'US', 'DEFAULT') GEO_ADDR
FROM DUAL;
```

```sql
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(NULL), NULL, 'CLAY ST', NULL, NULL, 'SAN FRANCISCO', NULL, 'CA', 'US', '94109', NULL, '94109', NULL, '1698', 'CLAY', 'ST', 'F', 'F', NULL, NULL, 'L', 0, 23600689, '???? #ENUT?B281CP?', 1, 'DEFAULT', -122.42093, 37.79236, '????4101010?? 004?'))
```

MATCHCODE 1 = exact match with provided input
Match on the center house number of Clay St
Postal code filled in, street address corrected
Example: House Level Match

SELECT SDO_GCDR.GEOCODE('SPATIAL',
    SDO_KEYWORDARRAY('1350 Clay', 'San Francisco, CA'),
    'US', 'DEFAULT') GEO_ADDR
FROM DUAL;

MATCHCODE 2 = street type not matched
Match on the exact house number of Clay St
Postal code filled in, street address completed
Example: Address Correction

```sql
SELECT SDO_GCDR.GEOCODE('SPATIAL',
    SDO_KEYWORDARRAY('1350 Clay Avenue', 'San Francisco, 94119 CA'),
    'US', 'DEFAULT') GEO_ADDR
FROM DUAL;
```

MATCHCODE 10 = postal code not matched
Match on the exact house number of Clay St
Postal code and street name corrected
Example: Address Spelling Correction

```
SELECT SDO_GCDR.GEOCODE('SPATIAL',
    SDO_KEYWORDARRAY('100 VAN NOUSS', 'San Francisco, CA'),
    'US', 'DEFAULT') GEO_ADDR
FROM DUAL;
```

```
    '????0121410???004?')
```

MATCHCODE 4 = street address not matched
Fuzzy match on street name
Postal code inserted and street name corrected
Multiple matches

The `geocode_all()` function

```sql
SELECT SDO_GCDR.GEOCODE_ALL('SPATIAL',
    SDO_KEYWORDARRAY('Clay Street', 'San Francisco, CA'),
    'US', 'DEFAULT') GEO_ADDR
FROM DUAL;

SDO_ADDR_ARRAY (
  SDO_GEO_ADDR(1, SDO_KEYWORDARRAY(), NULL, 'CLAY ST', NULL, NULL,
    'SAN FRANCISCO', NULL, 'CA', 'US', '94109', NULL, '94109', NULL, '1698',
    'CLAY', 'ST', 'F', 'F', NULL, NULL, 'L', 0, 23600700, '????#ENUT?B281CP?',
    1, 'DEFAULT', -122.42093, 37.79236, '????4101010??004?'),
  SDO_GEO_ADDR(1, SDO_KEYWORDARRAY(), NULL, 'CLAY ST', NULL, NULL,
    'SAN FRANCISCO', NULL, 'CA', 'US', '94108', NULL, '94108', NULL, '978',
    'CLAY', 'ST', 'F', 'F', NULL, NULL, 'L', 0, 23600689, '????#ENUT?B281CP?',
    1, 'DEFAULT', -122.40904, 37.79385, '????4101010??004?'),
  ...
)
```
Simple interface: the `geocode_as_geometry()`

```sql
SELECT SDO_GCDR.GEOCODE_AS_GEOMETRY('SPATIAL',
    SDO_KEYWORDARRAY('1350 Clay', 'San Francisco, CA'), 'US')
FROM DUAL;
```

```sql
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.41522, 37.7930729, NULL), NULL, NULL)
```

No match_mode parameter: always DEFAULT mode
No matchcode returned
Using Structured Input

Pass individual address fields in a structured way

• **SDO_GCDR.GEOCODE_ADDR**
  • Like GEOCODE, but takes an SDO_GEO_ADDR object as input

• **SDO_GCDR.GEOCODE_ADDR_ALL**
  • Like GEOCODE_ALL, but takes an SDO_GEO_ADDR object as input

• Useful when input data is already structured
  • Avoids unnecessary address parsing and parsing errors
Structured Geocoding functions

```sql
SDO_GEO_ADDR = SDO_GCDR.GEOCODE_ADDR (  
  <USER_NAME>, <SDO_GEO_ADDR>)

SDO_GEO_ADDR_ARRAY = SDO_GCDR.GEOCODE_ADDR_ALL (  
  <USER_NAME>, <SDO_GEO_ADDR>)
```
SELECT SDO_GCDR.GEOCODE_ADDR('SPATIAL',
    SDO_GEO_ADDR (  
        'US',              -- COUNTRY
        'DEFAULT',          -- MATCHMODE
        '1200 Clay Street', -- STREET
        'San Francisco',    -- SETTLEMENT
        NULL,               -- MUNICIPALITY
        'CA',               -- REGION
        '94108'             -- POSTALCODE
    )
) FROM DUAL;
Point Address Geocoding

- Point Addressing data has an exact long/lat for each address
  - This is different from the range based addressing where each road segment has an address range
- Support for this feature requires a new data table in addition to the current set of Geocoder tables
- New table: GC_ADDRESS_POINT_NVT
- No interface change required, if this table exists we use it to refine the result using the exact long/lat provided in the table
Point Address Geocoding Data

- Customers can buy this new data set from NAVTEQ
- NAVTEQ Point Addressing data product
- This is in addition to the NAVTEQ Geocoding data in Oracle Data Format (ODF)
**Example: Point Address**

```
SELECT SDO_GCDR.GEOCODE('SPATIAL',
    SDO_KEYWORDARRAY('1000 Howard st', 'San Francisco, CA'),
    'US', 'DEFAULT') GEO_ADDR
FROM DUAL;
```

```
```

**MATCHCODE 1 = exact match**

Match on the exact house number of Clay St
Postal code filled in, street address completed
Point address match found
Reverse Geocoding

• This is the opposite of geocoding
• Given a point location, determine the street address
  • Find the nearest road segment
  • Project the point location on the road segment
  • Compute house number by interpolation
  • Fill in address details: street name, postal code, city name, etc.
Reverse Geocoding Process

**Geocoder**
- Find Road Segment
- Compute House Number
- Fill Address Details

**Reference Data for Geocoding**
- Geometries
- Geometries
- Street and place names

**Location Coordinates**

**Address**
Reverse Geocoding Example

SELECT SDO_GCDR.REVERSE_GEOCODE ('SPATIAL',
   SDO_GEOMETRY(2001, 8307,
       SDO_POINT_TYPE(-122.41522, 37.7930729, NULL), NULL, NULL),
   'US')
FROM DUAL;


- Note that the coordinates returned are on the road segment
- Can be different from the input coordinates
What if you do not like objects?

- Use a sub-query to extract results from SDO_GEO_ADDR

```sql
SELECT G.GC.STREETNAME, G.GC.HOUSENUMBER, G.GC.POSTALCODE, 
       G.GC.SETTLEMENT 
FROM ( 
    SELECT SDO_GCDR.GEOCODE ('SCOTT', 
                            SDO_GEOMETRY(2001, 8307, 
                                   SDO_POINT_TYPE(-122.41522, 37.7930729, NULL), NULL, NULL), 
                            'US') GC 
    FROM DUAL) G; 
```

<table>
<thead>
<tr>
<th>GC.STREETNAME</th>
<th>GC.H</th>
<th>GC.PO</th>
<th>GC.SETTLEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAY ST</td>
<td>1350</td>
<td>94109</td>
<td>SAN FRANCISCO</td>
</tr>
</tbody>
</table>
What if you do not like objects?

- Can also use the WITH clause

```sql
WITH GC_RESULT AS (
    SELECT SDO_GCDR.REVERSE_GEOCODE ('SPATIAL',
        SDO_GEOMETRY(2001, 8307,
            SDO_POINT_TYPE(-122.41522, 37.7930729, NULL), NULL, NULL),
        'US') GC
    FROM DUAL
)
SELECT GC.STREETNAME, GC.HOUSENUMBER, GC.POSTALCODE,
    GC.SETTLEMENT
FROM GC_RESULT GC;
```

<table>
<thead>
<tr>
<th>GC.STREETNAME</th>
<th>GC.H</th>
<th>GC.PO</th>
<th>GC.SETTLEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAY ST</td>
<td>1350</td>
<td>94109</td>
<td>SAN FRANCISCO</td>
</tr>
</tbody>
</table>
What if you do not like objects?

- Use the TABLE() notation with a GEOCODE_ALL call

```sql
SELECT STREETNAME, HOUSENUMBER, SETTLEMENT, LONGITUDE, LATITUDE
FROM TABLE (SDO_GCDR.GEOCODE_ALL('SPATIAL',
  SDO_KEYWORDARRAY('1350 Clay', 'San Francisco, CA'),
  'US', 'DEFAULT'))
```

<table>
<thead>
<tr>
<th>STREETNAME</th>
<th>HOUSENUMBER</th>
<th>SETTLEMENT</th>
<th>LONGITUDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLAY ST</td>
<td>1350</td>
<td>SAN FRANCISCO</td>
<td>-122.41522</td>
</tr>
<tr>
<td>37.7930729</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLAYTON ST</td>
<td>1350</td>
<td>SAN FRANCISCO</td>
<td>-122.44639</td>
</tr>
<tr>
<td>37.7593208</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Installing and configuring the geocoding web service

- Deploy GEOCODER.EAR using the application server console
  - File is in $ORACLE_HOME/md/jlib
  - The geocoder startup fails because the default configuration points to an inaccessible database
- Update geocoder configuration
  - Using geocoder console: http://<server>/geocoder/admin.jsp
  - Or manually update file $J2EE_HOME/applications/geocoder/web/WEB-INF/config/geocodercfg.xml
  - Set proper database connection to database user that owns the geocoding tables
Using the Geocoding Web Service

Request:

```xml
<?xml version="1.0" standalone="yes"?>
<geocode_request>
  <address_list>
    <input_location id="1">
      <input_address match_mode="default">
        <unformatted country="FR">
          <address_line value="13 rue Royale" />
          <address_line value="Paris" />
        </unformatted>
      </input_address>
    </input_location>
  </address_list>
</geocode_request>
```

Response:

```xml
<geocode_response>
  <geocode id="1" match_count="1">
    <match sequence="0" longitude="2.322855" latitude="48.867909999999995"
      match_code="1" error_message="????#ENUT?B281CP?" match_vector="????0101010??404?">
      <output_address name="" house_number="13" street="RUE ROYALE" builtup_area="PARIS"
        order1_area="ÎLE-DE-FRANCE" order8_area="" country="FR" postal_code="75008"
        postal_addon_code="" side="L" percent="0.75" edge_id="228533511"/>
    </match>
  </geocode>
</geocode_response>
```
Using a Structured Input Address

Request:

<?xml version="1.0" standalone="yes"?>
<geocode_request>
  <address_list>
    <input_location id="1">
      <input_address match_mode="default">
        <gen_form country="FR"
            street="13 rue Royale"
            city="Paris"
            postal_code="75008"
        />
      </input_address>
    </input_location>
  </address_list>
</geocode_request>

Response:

<geocode_response>
  <geocode id="1" match_count="1">
    <match sequence="0" longitude="2.322855" latitude="48.867909999999995" match_code="1" error_message="????#ENUT?B281CP?" match_vector="????0101010??404?">
      <output_address name="" house_number="13" street="RUE ROYALE" builtup_area="PARIS"
          order1_area="ÎLE-DE-FRANCE" order8_area="" country="FR" postal_code="75008"
          postal_addon_code="" side="L" percent="0.75" edge_id="228533511"/>
    </match>
  </geocode>
</geocode_response>
Reverse Geocoding

Request:

```xml
<?xml version="1.0" standalone="yes"?>
<geocode_request vendor="elocation">
  <address_list>
    <input_location id="1"
      country="FR"
      longitude="2.322855"
      latitude="48.86790999999959">
    </input_location>
  </address_list>
</geocode_request>
```

Response:

```xml
<geocode_response>
  <geocode id="1" match_count="1">
    <match sequence="0" longitude="2.3228550000005823" latitude="48.867909999999576"
      match_code="1" error_message="" match_vector="????4141414??404?">
      <output_address name="" house_number="13" street="RUE ROYALE" built-up_area="PARIS"
        order1_area="ÎLE-DE-FRANCE" order8_area="" country="FR" postal_code="75008"
        postal_addon_code="" side="L" percent="0.749999999984763" edge_id="228533511"/>
    </match>
  </geocode>
</geocode_response>
```
Reverse geocoding with projected point

Request:

```xml
<?xml version="1.0" standalone="yes"?>
<geocode_request vendor="elocation">
  <address_list>
    <input_location id="1"
        country="FR"
        x="598997.994"
        y="2429954.32"
        srid="41014">
      </input_location>
    </address_list>
  </geocode_request>
```

Response:

```xml
<geocode_response>
  <geocode id="1" match_count="1">
    <match sequence="0" longitude="2.32315" latitude="48.86832"
        match_code="1" error_message="" match_vector="????4141414??404??"
        <output_address name="" house_number="17" street="RUE ROYALE" builtup_area="PARIS"
            order1_area="ÎLE-DE-FRANCE" order8_area="" country="FR" postal_code="75008"
            postal_addon_code="" side="L" percent="1.0" edge_id="228533511"/>
    </match>
  </geocode>
</geocode_response>
```
Routing Engine
Routing Server

- Architecture
- Routing tables
- Partitioning the network
- Installation and configuration
- How to formulate route requests
- Structure of the route responses
The Oracle Spatial Routing Engine

• The Oracle Spatial Routing Engine is an XML-based Web service.
• Provides driving directions between two locations
  • Directions, estimated distance and drive time, geometry, ...
• Requires a suitable road network as well as geocoding data
• Users must purchase data from Oracle Partners to use the Spatial routing and geocoding functions
  • Data available from NAVTEQ
  • Other providers: format the data to meet the data model of the router
• Sample data is provided at Oracle partner sites. Links are provided from Spatial OTN web site:
  http://www.oracle.com/technology/products/spatial/htdocs/spatial_partners_data.html
Routing Tables

- **EDGE table**
  - Describes all road segments in the network
  - Each segment has a unique identifier
- **NODE table**
  - Describes all road intersections
- **SIGN_POST table**
  - Optional
  - Relates edges at intersections
  - Used for signposts at highway exits
- **PARTITION table**
  - One row per network partition
Installation and configuration

• Deploy ROUTESERVER.EAR using the application server console
  • The router startup fails because the default configuration points to an inaccessible database

• Update router configuration
  • Manually update file $J2EE_HOME/applications/routeserver/web/WEB-INF/web.xml
  • Set connection to user for routing tables
  • Choose type of geocoder to use and configure it
    • Database or web service
  • Set default language, default driving side, etc.

• Restart OC4J container!
Router Configuration

- **Type of geocoder to use**
  - geocoder_type ("thinclient", "httpclient" or "none")
  - geocoder_match_mode

- **Database connection for geocoding tables ("thinclient" geocoder)**
  - geocoder_schema_host
  - geocoder_schema_port
  - geocoder_schema_sid
  - geocoder_schema_username
  - geocoder_schema_password

- **URL to geocoding service ("httpclient" geocoder)**
  - geocoder_http_url
  - geocoder_http_proxy_host
Router Configuration

• **General parameters**
  - `local_road_threshold`
  - `highway_cost_multiplier`
  - `max_speed_limit`
  - `driving_side`
  - `language`
  - `distance_function_type`
A route request consists of:
- Preferences
- Start location
- End location

A batch route request consists of:
- Preferences
- Start location
- End locations

A route response consists of:
- Route information
- Optional Geometry
- Segment information (for each segment of the route)

A batch route response consists of:
- Route information (for each route)
XML Input for a Route Request

- Start location is an unformatted address
- End location is a formatted US address
A Route Request

An XML request for a single route includes:

- A numeric route ID
- One start and one end location
- Optionally, one or more attributes:
  - Route preference: fastest or shortest
  - Road preference: highway or local
  - Whether to return the geometry of the route
  - Time and distance units
  - Language for driving directions
## Route Request Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>route_preference</td>
<td>FASTEST SHORTEST</td>
</tr>
<tr>
<td>road_preference</td>
<td>HIGHWAY LOCAL</td>
</tr>
<tr>
<td>return_driving_directions</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>driving_directions_detail</td>
<td>LOW MEDIUM HIGH</td>
</tr>
<tr>
<td>return_hierarchical_driving_directions</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>return_route_geometry</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>return_segment_geometry</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>return_detailed_geometry</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>return_route_edge_ids</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>return_segment_edge_ids</td>
<td>TRUE or FALSE</td>
</tr>
<tr>
<td>language</td>
<td>ENGLISH FRENCH GERMAN ITALIAN SPANISH</td>
</tr>
<tr>
<td>distance_unit</td>
<td>KM MILE METER</td>
</tr>
<tr>
<td>time_unit</td>
<td>HOUR MINUTE SECOND</td>
</tr>
</tbody>
</table>
<?xml version="1.0" ?>
<route_response>
  <route
    id="8"
    step_count="14"
    distance="30.28667355371901"
    distance_unit="mile"
    time="35.02037760416667"
    time_unit="minute">
    <segment
      sequence="1"
      instruction="Start out on WINTER ST (Going South)"
      distance="1.2041612436793172"/>
    <segment
      sequence="2"
      instruction="Stay STRAIGHT to go onto TOTTEN POND RD (Going East)"
      distance="0.08879983757738225"/>
    . . . . .
    <segment
      sequence="14"
      instruction="Turn LEFT onto WYMAN ST (Going North)"
      distance="0.2468156965686923"/>
  </route>
</route_response>
Origin and destination

- The previous route request used addresses for the start and end locations.
- Can also specify a pre-geocoded location
  - Link id, side, relative position (0.0 to 1.0)
- Can also specify spatial coordinates
  - This is useful for Telematics and other applications.
- From a location specified by coordinates, what is the route to a particular address?
- Given a current location, what is the route to a location specified by coordinates?
<?xml version="1.0" standalone="yes"?>
<route_request id="1" route_preference="fastest"
   return_driving_directions="true"
   distance_unit="mile" time_unit="minute">
   <start_location>
      <input_location id="1"
         longitude="-122.4014128" latitude="37.7841193" />
   </start_location>
   <end_location>
      <input_location id="2">
         <input_address>
            <unformatted country="US" >
               <address_line value="1300 Columbus" />
               <address_line value="San Francisco, CA" />
            </unformatted>
         </input_address>
      </input_location>
   </end_location>
</route_request>
An XML request for a batch route includes:

- A numeric route ID
- One start and one or more end locations
- Optionally, one or more of the following attributes:
  - route_preference: Fastest (default) or shortest
  - road_preference: Highway (default) or local
  - distance_unit: Kilometer, mile (default), meter
  - time_unit: Hour, minute (default), second
  - sort_by_distance: Sorts the returned information in ascending order by distance (FALSE by default)
  - cutoff_distance: Returns information only if the distance is less than or equal to a specified distance (no limit by default)
<?xml version="1.0" standalone="yes"?>
<batch_route_request
    id="8"
    route_preference="fastest"
    road_preference="highway"
    return_driving_directions="false"
    sort_by_distance = "true"
    cutoff_distance="35"
    distance_unit="mile"
    time_unit="minute">
    <start_location>
        <input_location id="1">
            <input_address>
                <us_form1
                    street="1000 Winter St"
                    lastline="Waltham, MA" />
            </input_address>
        </input_location>
    </start_location>
    . . . .
    <end_location>
        <input_location id="10">
            <input_address>
                <us_form1
                    street="1 Oracle Dr"
                    lastline="Nashua, NH" />
            </input_address>
        </input_location>
    </end_location>
    . . . .
    <end_location>
        <input_location id="11">
            <input_address>
                <us_form1
                    street="22 Monument Sq"
                    lastline="Concord, MA" />
            </input_address>
        </input_location>
    </end_location>
</batch_route_request>
XML from a Batch Route Request

- One <route> element per destination
- Includes distance and time
- Destinations ordered by distance

```xml
<?xml version="1.0" standalone="yes" ?>
<batch_route_response id="8">
  <route id="11"
    step_count="0"
    distance="9.132561517429938"
    distance_unit="mile"
    time="12.4705078125"
    time_unit="minute" />
  <route id="12"
    step_count="0"
    distance="17.74747391140558"
    distance_unit="mile"
    time="20.413236490885417"
    time_unit="minute" />
  <route id="10"
    step_count="0"
    distance="30.28667355371901"
    distance_unit="mile"
    time="35.02037760416667"
    time_unit="minute" />
</batch_route_response>
```
Routing Engine

- Routing Application is now built on top of NDM LOD engine
- Routing Engine uses all the extensibility of the underlying NDM engine
- Truck routing is an example of customizing the routing engine specific to an application based on user data
Truck Routing

- `<?xml version="1.0" standalone="yes"?> <route_request id="8" route_preference="shortest" road_preference="highway" vehicle_type="truck" return_driving_directions="true" distance_unit="mile" time_unit="hour" return_route_geometry="false" >`

- Route Request `vehicle_type`
  - (auto|truck) optional, defaults to auto

- Route Request `truck_type`
  - `truck_type`: (delivery|public|trailer) optional, no default
• truck_height: (positive float) optional, no default
• truck_length: (positive float) optional, no default
• truck_per_axle_weight: (positive float) optional, no default
• truck_weight: (positive float) optional, no default
• truck_width: (positive float) optional, no default
• length_unit: (metric|us) optional, default US
• weight_unit: (metric|us) optional, default US
• Truck height, length and width are specified in length_unit units
• Truck per axle weight and weight and specified in weight_unit units
Truck Routing Data

• Table definition for trucking data table
• This table contains data from the NVT_TRANSPORT table from NAVTEQ
  • NAVTEQ Transport Product
• Trucking User Data table

ROUTER_TRUCK_DATA (edge_id NUMBER, maintype NUMBER(2), subtype NUMBER(2), value NUMBER(2));
Demonstration Title Here
Javascript APIs for Geocoder and Routing Engine
Using the new eLocation GeoCoding & Routing APIs

• A new JavaScript class: `OracleELocation`
• Two methods:
  • for geocoding: `geocode(address, callBack, errHandler)`
  • for routing/directions: `getDirections(…)`
How to import the new API scripts

In your HTML page, import the API scripts from elocation.oracle.com:

```html
<script src="http://elocation.oracle.com/elocation/jslib/oracleelocation.js"
type="text/javascript">
</script>
```

You must also import the MapViewer JS API:

```html
<script src="http://elocation.oracle.com/mapviewer/fsmc/jslib/oraclemaps.js" type="text/javascript">
</script>
```

You are now ready to make use of the new OracleELocation class.

The online API documentation can be found here:

http://elocation.oracle.com/elocation/ajax/apidoc/symbols/OracleELocation.html
Construct a new OracleELocation instance

After importing the class scripts, you will create a new OracleELocation instance.

```javascript
<script type="text/javascript">
  var eloc;
  ...

  function init()
  {
    eloc = new OracleELocation("http://elocation.oracle.com/elocation");
  }
</script>
```

The constructor of OracleELocation takes as a parameter the URL to the eLocation server.
How to geocode an address

To geocode a street address (US and Western EU countries only at the moment), simply invoke the geocode() method on the OracleELocation object just created.

```javascript
var myAddress = "500 Oracle Pkwy, Redwood Shores, CA, 94065";
eloc.geocode(myAddress, printResult);
```

The “geocode” function internally invokes an AJAX call to the Oracle eLocation service for the actual work.

Callback function:

Note the function named “printResult”. It is a user-supplied callback function that will be invoked automatically when the geocoding result is returned from the eLocation server. It is inside this callback function that you will interpret the geocoding result and do something with it, such as printing the longitude/latitude of the geocoded address, or display it on a map.
How to interpret the geocoding result

In your callback function, the geocoding result is passed in as the only argument.

```javascript
function printResult(gcResult)
{
    //interpret and process the geocoding result
    ...
}
```

A few things worth noting about the `gcResult` parameter:
- It is an array of (geocoded/validated) addresses
- If the input address has multiple matches, gcResult will have multiple address objects
- Each address object contains the following attributes:
  - x: The x (longitude) coordinate of the result location.
  - y: The y (latitude) coordinate of the result location.
  - houseNumber: Address street house number
  - street: Street
  - settlement: City
  - municipality: Municipality
  - region: Region(state, province, etc.)
  - postalCode: Postal code
  - country: Country
  - matchVector: Match vector that tells how each address field is matched. Please refer to Oracle Spatial Developer's guide for more information.
  - accuracy: Result accuracy
  - matchCode: Match code
The following is a complete geocoding example.

```html
<html>
<head>
<title>Oracle eLocation API Example: Simple Geocoding</title>
<script src="http://elocation.oracle.com/elocation/jslib/oracleelocation.js" type="text/javascript"></script>
<script src="http://elocation.oracle.com/mapviewer/fsmc/jslib/oraclemaps.js" type="text/javascript"></script>
<script type="text/javascript">
var eloc;

function init() {
    eloc = new OracleELocation("http://elocation.oracle.com/elocation");
}

function submitAddress() {
    var addr = document.getElementById("address").value
    eloc.geocode(addr, printResult);
}

function printResult(gcResult) {
    if(gcResult.length==0) //no match
        alert("No matching address found!");
    else if(gcResult.length==1) //1 exact match
        printJustOne(gcResult[0]);
    else
        alert("Multiple matches found. Printing 1st matched address only.");
}

function printJustOne(gcResult) {
    if(gcResult.length==0) //no match
        alert("No matching address found!");
    else if(gcResult.length==1) //1 exact match
        printJustOne(gcResult[0]);
    else
        alert("Multiple matches found. Printing 1st matched address only.");
}
</script>
</head>
<body onload="init()">
<input type="text" name="address" id="address" size=60 value="500 Oracle pkwy, Redwood Shores, CA 94065" />
<input type="button" onclick="submitAddress()" value="Geocode this..."/>
</body>
</html>
```
How to get driving directions

The new eLocation class also provides a convenient method for getting driving directions between two or more addresses.

```javascript
var addr = new Array();
addr[0] = "500 Oracle Pkwy, Redwood City, CA, 94065";
addr[1] = "170 West Tasman Dr., San Jose, CA 95134";
addr[2] = "345 Park Avenue, San Jose, CA 95110";

eloc.getDirections(addr, printResult, routeErrHandler,
  {routePref:"shortest", roadPref:"highway", distUnit:"km"},
  {mapview:mapview, zoomToFit:true, resultPanel:document.getElementById("directions")});
```

As you can see, getDirections() is a very powerful method. It can take two or more addresses (stops), and produce a route for each consecutive pair of addresses, complete with turn-by-turn directions. You can also specify the routing preferences. Finally you can display the route result both graphically on an Oracle Maps map, and in a tabular form for the turn-by-turn directions.
How to get driving directions – ctd.

The getDirections() method takes the following parameters:

- **dests** - an array of destination addresses (except the 1st one which is the starting address)
- **callBack** – a user-supplied callback function for custom route result processing
- **errHandler** – a user-supplied error handling function; invoked automatically if the routes cannot be computed (such as when one of the input addresses is invalid.)
- **routeOptions** – an object that specifies various routing preferences
- **mapOptions** – an object that specifies where and how the routes can be displayed on a map

Note that when displaying routes on a map, the map display area must reside on the same Web page where the routing code is invoked.

Note also if you use a custom callback function to display or process the route result, then the mapOptions object should be set to null.

For more details on the attributes of the routeOptions and mapOptions objects, please refer to the online API doc here:

http://elocation.oracle.com/elocation/ajax/apidoc/symbols/OracleELocation.html
How to get driving directions - a screenshot

eLocation AJAX Routing API demo

Estimated time: 36 min
Distance: 47.7 km

<table>
<thead>
<tr>
<th>Direction</th>
<th>Distance</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start from 500 ORACLE PKY, REDWOOD CITY, CA 94065</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start out on ORACLE PKY (Going West)</td>
<td>912 m</td>
<td>1 min</td>
</tr>
<tr>
<td>Turn RIGHT onto MARINE PKY/MARINE WORLD PKY (Going West)</td>
<td>135 m</td>
<td>6 sec</td>
</tr>
<tr>
<td>Stay STRAIGHT to go onto RALSTON AVE (Going Southwest)</td>
<td>308 m</td>
<td>21 sec</td>
</tr>
<tr>
<td>Take US-101 AVE  (Going Southwest)</td>
<td>853 m</td>
<td>41 sec</td>
</tr>
<tr>
<td>Stay STRAIGHT to go onto RAMP (Going Southeast)</td>
<td>469 m</td>
<td>20 sec</td>
</tr>
<tr>
<td>Stay STRAIGHT to go onto US-101 S (Going Southeast)</td>
<td>24.8 km</td>
<td>13 min</td>
</tr>
<tr>
<td>Take CA-237 RAMP toward ALVISO</td>
<td>406 m</td>
<td>37 sec</td>
</tr>
<tr>
<td>Stay STRAIGHT to go onto CA-101 E (Going East)</td>
<td>489 m</td>
<td>19 sec</td>
</tr>
</tbody>
</table>
April 2010
Oracle Spatial
User Conference