Oracle® Transportation Management

Data Management Guide
Release 6.3
Part No. E38426-07

April 2015

Oracle Transportation Management Data Management Guide, Version 6.3

Part No. E38426-07

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Contents

23. APPENDIX B – JAVA INTEGRATION API (DEPRECATED)............... 23-1

Send Us Your Comments

Oracle Transportation Management Data Management Guide, Release 6.3

Part No. E38426-07

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

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

- Electronic mail: otm-doc_us@oracle.com

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

If you have problems with the software, contact Support at https://support.oracle.com or find the Support phone number for your region at http://www.oracle.com/support/contact.html.

Preface

This manual is for members of the Oracle Transportation Management implementation team, who are responsible for maintaining and updating data in Oracle Transportation Management at your site. This manual provides step-by-step instructions for importing and exporting data in CSV and db.xml format.

This manual does not cover the installation of any components required to import or export. See the Administration Guide for for installation and configuration instructions. The latest version of the guide can be found on the OTN website.

> **Note**: This manual provides examples of CSV, XML and schema diagrams. For actual database tables and schema, refer to the latest database schema and the GlogXML schema.

Change History

Date	**Document Revision**	**Summary of Changes**
11/2012	-01	Initial release.
03/2013	-02	Rewrote chapter 3. Explained Command Line utilities. Consolidated chapter 3 and 4. Replaced chapter 26. Removed reference to glog.integration.clientapi.CSVHelper. Updated descriptions of DB XML import and export. Moved all references regarding Python to new appendix called "Using Python".
08/2013	-03	Reinstate "Process Rate Factor" section based on new Java Command Line Processor. New Migration Project feature.
09/2013	-04	Removed references to Select Lists.
12/2013	-05	Enhancements to Migration Project Moved Oracle Advanced Queue section to Integration Guide Deprecate Java Integration API
09/2014	-06	Enhancements to Migration Project Export Import Support for EMAIL and FTP of Exported Package File Support for Local uploading of Package File for Import
04/2015	-07	Corrected instruction in step 7 of chapter 15 that was out of order.

1. Introduction

DB.XML

DB.XML (Database-centric XML) is an XML file format for importing and exporting Oracle Transportation Management data.

The DB XML tool facilitates the direct query/update of data directly from/to the OTM database tables. As such, the tool should only be used by those already familiar with the responsibilities and capabilities that come with using such tools and who may already be familiar with database tools like SQLDeveloper, TOAD etc.

> **NOTE:** Updates made directly to the OTM database by DB XML Import can only ensure data consistency with respect to the standard database constraints, e.g. Primary Key, Foreign Key, and Check constraints. Imports do not flow through the main application logic for updates, and so cannot check that the business context of a particular change makes sense. For example, the status of a particular object (e.g. LOCATION STATUS) can be updated. Import can only check that the status GID is valid but not that the status, possibly in association with other status values, constitutes an appropriate state for the object to be in.

In the DB XML file, there can be more than one element contained within what is called a **Transaction Set**. The `TRANSACTION_SET` element is used to contain these **parent** elements. The parent element itself may contain one or more child element. DB XML Import and Export can work with complete parent-child table relationships all in one file by using corresponding parent-child elements. The attribute values on each element correspond to column values.

> **Note:** The convention used here is that a table is called the "child" table if it contains a foreign key to another table. The table referenced by the foreign key is called the "parent".

These parent elements typically correspond to the primary OTM data objects – AGENT, LOCATION, etc., and child elements typically correspond to associated child tables, For example, for the LOCATION parent table, the child table could be LOCATION_CORPORATION, LOCATION_REFNUM, etc..

In the case where the transaction set is used for data import, each parent element will, by default, be treated as a distinct transaction, i.e. the parent element and all its child elements are saved to the database as one atomic transaction. If one child element fails, the parent element transaction fails. The failure of one parent element does not directly affect the transactions for other parent elements. Additionally, all parent elements can be treated as one unit of work i.e. if one element fails, ALL elements in the 'set' will fail. The details on how this is achieved are covered in section 2, DB.XML.

Oracle Transportation Management ignores element and attribute names that do not correspond to valid database table or column names. This allows you to comment your DB.XML file without affecting what is imported.

Why do I want to use DB.XML?

Compared to CSV (Comma Separated Values), DB.XML supports manipulation of parent-child records as a unit. This gives DB.XML an advantage compared to CSV when updating, for example, rate information.

How can I use DB.XML?

There are a few ways to perform a DB.XML export or import:

- OTM User Interface

- Command Line: can be run directly connected to the database (when local SQL*Net connection is available) or by using the OTM web or application server remotely.
- HTTP POST to servlet on OTM Web server (requires authentication)
- SOAP web service

See section 2, DB.XML for details.

CSV

CSVUtil is a utility for importing and exporting data in CSV format in and out of the Oracle Transportation Management database. CSVUtil also exports data as a script of insert statements. This document describes how to use CSVUtil and shows some sample CSV files.

CSV files are compact and enable you to import large amounts of data into Oracle Transportation Management. You typically want to use CSVUtil when importing rates into a fresh installation of Oracle Transportation Management.

There are three ways to use CSVUtil:

- On the DOS/UNIX command line
- Via the Oracle Transportation Management web interface
- Via integration transmissions

A Sample CSV File

Below is a sample CSV file:

```
ICON
ICON_GID,ICON_XID,DESCRIPTION,PATH,DOMAIN_NAME,INSERT_USER,INSERT_DATE,UPDATE_USER,UPDATE_DATE
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
"BATCH_GRID","BATCH_GRID","Reports Batch Grid","/images/icons/reports/batch_grid.jpg","PUBLIC","DBA.ADMIN","20040310091645","DBA.ADMIN","20040630100834"
```

Line 1 must be the name of the table.

Line 2 must be a comma-separated list of column names. Only the columns being loaded must be specified.

After line 3 may be one or more optional EXEC SQL lines, such as the one shown above, to set the date format.

Subsequent lines include the data. The number of columns of data must correspond to the number of columns specified on line 2. The ordering of the data columns must also correspond to line 2.

Character data may be surrounded with double-quotes, as shown above. If you need to include a double-quote character, use “&quot;” instead. The tools described here to export CSV files automatically convert double-quote characters into “&quot;”.

Numeric data should not be surrounded with double-quotes.

Multi-table CSV Files

The output produced by the xcsvw* commands is in multi-table CSV format. The various CSV import commands recognize this format also

The first record in a multi-format file must be "$HEADER".

The header section contains table names and the names of the columns used in that table.

After the header section comes the body, identified by the $BODY keyword.

Each data record in the $BODY must be preceded by its table name on the prior line.

Here is an example:

```
$HEADER
LOCATION_ROLE_PROFILE
LOCATION_GID,LOCATION_ROLE_GID,CALENDAR_GID,FIXED_STOP_TIME, etc...
LOCATION_STATUS
LOCATION_GID,STATUS_TYPE_GID,STATUS_VALUE_GID,DOMAIN_NAME,INSERT_USER,INSERT_DA
TE,UPDATE_USER,UPDATE_DATE
LOCATION_CORPORATION
LOCATION_GID,CORPORATION_GID,DOMAIN_NAME,INSERT_DATE,UPDATE_DATE,INSERT_USER,UP
DATE_USER
LOCATION_ADDRESS
LOCATION_GID,LINE_SEQUENCE,ADDRESS_LINE,DOMAIN_NAME,INSERT_USER,INSERT_DATE,UPD
ATE_USER,UPDATE_DATE
LOCATION_REFNUM
LOCATION_GID,LOCATION_REFNUM_QUAL_GID,LOCATION_REFNUM_VALUE,DOMAIN_NAME,INSERT_
DATE, etc...
LOCATION
LOCATION_GID,LOCATION_XID,LOCATION_NAME,ADDRESS_LINE1,ADDRESS_LINE2,CITY,etc.
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS..'
$BODY
LOCATION
"GUEST.00621918","00621918","00621918",,,,,"TN",,"USA",,,,,"America/New_York",,
,,,,,,"N","N","COMMERCIAL",,,"GUEST","S",0,...etc
LOCATION_ADDRESS
"GUEST.00621918",1,,"GUEST","DBA.ADMIN",2001-10-07 17:53:53.0,,
LOCATION_ADDRESS
"GUEST.00621918",2,,"GUEST","DBA.ADMIN",2001-10-07 17:53:53.0,,
LOCATION_CORPORATION
"GUEST.00621918","GUEST.CUST NO","GUEST",2001-10-15 10:50:49.0,,"DBA.ADMIN",
LOCATION_REFNUM
"GUEST.00621918","GLOG","GUEST.00621918","GUEST",2001-10-25 17:13:48.0,2001-10-
19 18:23:17.0,"DBA.ADMIN","DBA.GLOGOWNER"
LOCATION_ROLE_PROFILE
"GUEST.00621918","SHIPFROM/SHIPTO",,0,0,"GUEST","S",0,"S",0,"N",,,,,,,,,,2001-
10-25 14:12:38.0,2002-08-28 19:13:05.0,"DBA.ADMIN", etc.
LOCATION_STATUS
"GUEST.00621918","GUEST.CREDIT LEVEL","GUEST.CREDIT
LEVEL_UNKNOWN","GUEST","DBA.GLOGOWNER",2001-10-17 09:38:05.0,,
```

International Characters

Import

To be able to send data to Oracle Transportation Management containing characters outside the 7-bit ASCII character set, you must:

- Make sure your database uses an encoding that can handle all the characters you need.
- Always save your files using UTF-8 format.

XML Spy, Textpad and Notepad (Microsoft Windows 2000 or better) can all save in UTF-8 format.

Before you edit your files, you need to ensure that you configure your text editor to use the appropriate font and script (sometimes called subset). A script is a collection of characters such as Western European, Greek or Turkish. For example, if you need to update files containing Czech characters, then you need to select a font that supports an Eastern European script such as Arial or Arial Unicode Ms.

Export

When exporting files, Oracle Transportation Management writes files in UTF-8. Note that when you view data in your browser and then use the view source option to save your data, just save your file without specifying an encoding. Later, when editing your file, use an editor that support UTF-8.

Best Practices

Whether you are using DB XML or CSV export, you should follow some basic rules to help maintain proper version control and avoid data inconsistencies.

In order to maintain proper version control and track all changes that are being made to agents, we recommend that you **do not update existing active agents**. Instead, we recommend the current agent be disabled and that **a new agent is created** with the changes that need to be made. This will allow you to easily revert back to the previous version should you run into anything unexpected when the new agent is being used.

This can be achieved simply by adding a date, "02252013" or a version identifier ‘V#’, i.e. "V1”, “V2”, “V3" etc. to the end of the AGENT_XID when the agent is being created. If an existing agent is exported, modify the AGENT_GID and AGENT_XID before it is imported into another instance.

For instance if you want to create a new agent you may call it "SHIPMENT-CREATED V1".

If you decide you want to make changes to this agent you would create a new agent called "SHIPMENT-CREATED V2", turn off "SHIPMENT-CREATED V1", and turn on the new agent.

Now if you decide the new agent is not working as expected the new agent can simply be turned off and the original agent can be turned back on to restore the original workflow.

Migration Project

The Migration Project feature, added in Oracle Transportation Management v6.3.2, introduces a standard way to define and manage one or more datasets for the purpose of migrating data from one Oracle Transportation Management instance to another.

Although the Oracle Transportation Management application is fully functional “out of the box”, an operational system will typically require some custom configuration. Best practice would be for such a configuration to be developed and tested in a pre-production environment, accepted by product and business/operational experts and then promoted to the production environment.

The Migration Project is designed to facilitate this “promote to production” process which includes the initial release as well as subsequent incremental releases.

At a high level, a source system will be used to create a Migration Project Package by exporting an ‘Export’ Migration Project defined on that system. The package, essentially a ZIP file containing all the exported data, will then be transferred to the target system and used by an import process run on one of the target application servers. The import process will load the data contained in the package into the target system database. A corresponding ‘Import’ Migration Project will also be created on the target system to list the imported data and the success or failure of the import process.

2. DB.XML

DB XML Export

The DB XML Export process produces a **transaction set** which can be viewed in the UI or saved as an external XML file.

There are a number of ways to specify the data to be exported:

- Using an Object Name
- Using an Object Set Name
- Specifying a SQL Query
- Migration entity

For all methods described below there is an option to specify whether the so-called "foot print" columns are included in the export. The foot print columns are the INSERT_DATE, INSERT_USER, UPDATE_DATE, and UPDATE_USER. The values for these columns would be updated for any subsequently imported data by way of INSERT/UPDATE triggers and so their presence in exported data is largely informational.

Note: Starting from version 6.3.3, all exported DB.XML will be marked with a "Version" attribute specifying the version of the source OTM system.

There is a new option introduced in 6.3.3 to export Large Object column values (LOB) as elements rather than attributes. By default, LOB data (BLOB and CLOB) is exported as a 'base 64' encoded string attribute value. Working with the CLOB data, for example, would require decoding the string to obtain the original character text, editing the text and then encoding again to a 'base 64' string.

To make this process easier it is now possible to export LOB data as separate elements. For example, if `parent_col2` is a CLOB column and `parent_col3` is a BLOB column:-

```
<?xml version="1.0" encoding="UTF-8"?>
<xml2sql Version=”6.3.3”>
   <TRANSACTION_SET useLOBElement="true">
      <"parent table name" parent_col1="value_1" parent_col2="CLOBID#1"
parent_col3="BLOBID#1" ..etc..>
         <"child table name" child_col1="value_1" ..etc..>
            ...etc...
         </"child table name">
         <CLOB ID="CLOBID#1">
            <![CDATA[
               ...Both xml and non-xml text appears here...
            ]]>
         </CLOB>
         <BLOB ID="BLOBID#1">
            ...base 64 encoded binary value appears here...
         </BLOB>
      </"parent table name">
   </TRANSACTION_SET>
 </xml2sql>
```

The attribute value for the LOB column will then contain a unique ID for the element containing the CLOB or BLOB content. Note that BLOB content must always be a 'base 64' encoded string. The "useLOBElement" is generated on export and required on import for DB.XML files that contain the above format.

Using an Object Name

The **Object Name** is intended to refer to one of the OTM primary objects e.g. LOCATION, AGENT, RATE_GEO etc., though this is not enforced. The name actually corresponds to a pre-configured property which contains the name of file containing the SQL query to execute to retrieve the required data.

For example, the following property specifies the file to be used to retrieve data for the LOCATION Object Name:

```
glog.integration.dbxml.query.LOCATION=sql/Location.sql
```

> **Note:** the file location above is relative to the `<OTM_HOME>/glog/glog_resources` directory but it could be any file that is available on the CLASSPATH.

The following is an excerpt from the contents of the file provided with the standard installation:-

```
select location.*,
   cursor (select location_accessorial.* from location_accessorial where
location_accessorial.location_gid = location.location_gid) as
location_accessorial,
…etc…
from location
```

The use of a "cursor" is to produce the child element, in this case for the LOCATION_ACCESSORIAL records for this location, and there is a cursor for every required child table (and in "grandchild" tables and so on).

The base install will provide the following pre-configured queries:

Object Name	**SQL File**
LOCATION	sql/Location.sql
RATE_GEO	sql/RateGeo.sql
RATE_OFFERING	sql/RateOffering.sql
AGENT_ACTION	sql/AgentAction.sql
AGENT_EVENT	sql/AgentEvent.sql
AGENT	sql/Agent.sql
CORPORATION	sql/Corporation.sql
SAVED_QUERY	sql/SavedQuery.sql
SAVED_CONDITION	sql/SavedCondition.sql
USER_MENU_LAYOUT	sql/UserMenuLayout.sql
MONITOR_PROFILE	sql/MonitorProfile.sql
SHIPMENT	sql/Shipment.sql

Object Name	**SQL File**
STATUS_TYPE	sql/StatusType.sql
WORKFLOW_TOPIC_INFO	sql/WorkflowTopicInfo.sql
NOTIFY_SUBJECT_CONTACT	sql/NotifySubjectContact.sql
PLANNING_PARAMETER	sql/PlanningParameter.sql
BN_RULE	sql/BNRule.sql
NOTIFY_SUBJECT_STYLESHEET	sql/NotifySubjectStylesheet.sql
OB_ORDER_BASE	sql/ObOrderBase.sql

New custom object names and SQL files can be added by setting the associated properties.

The use of a ‘where clause’ is optional for the export by Object Name as it is feasible that the SQL file can contain the complete statement.

> **Note:** In order to be somewhat generic, the provided files all require a ‘where clause’ to retrieve a specific record. Without it, ALL records for the object name will be retrieved.

Using an Object Set Name

The **Object Set Name** is a named list of Object Names (described above). This allows a logical grouping of data to be exported in one file. The base install will provide the following pre-configured queries:

Object Set Name	**Associated Object Names**
DomainReferenceData	STATUS_TYPE, WORKFLOW_TOPIC_INFO, NOTIFY_SUBJECT_CONTACT, PLANNING_PARAMETER, BN_RULE, NOTIFY_SUBJECT_STYLESHEET

New custom object set names and object lists can be added by setting the associated properties.

Specifying a SQL Query

The complete SQL query (similar to the contents of the provided Object Name files) can also be specified directly. When this approach is used there must be an additional “Root Name” parameter given. This is used as the name of the top level parent element name for each record retrieved by the query.

Migration Entity

The Migration Entity names are a list of all the application objects available in OTM. They are designed to support the export of a top level object, e.g. Location and all its child objects, e.g. LocationStatus, LocationRefnum, etc. However, the list also contains the respective child entity names to support fine grained export.

The specific entities are retrieved based on a comma separated list of unique object primary keys for the entity name selected. For example, if Location were the selected entity the Object ID would be a comma separated list of Location Primary Key strings.

DB XML Import

The DB XML import process takes a **transaction set**, contained in an input XML file or message, and inserts, updates, or deletes rows in OTM tables. It can also completely replace a current set of child records with a new set.

Execution Mode

The execution of the import can be done in one of 3 execution modes:

- APP (default): Persistence is done via the application business logic
- SQL: Persistence is done via direct connection to the database
- PLSQL (Future use): Persistence is done via direct connection to the database but as one PL/SQL block execution. **Note:** as this is for future use this mode delegates to the SQL mode.

When the APP execution mode is used this makes available the capabilities to refresh associated application cache and to raise lifetime events for the objects create, modified or removed as part of the import. These capabilities are not available to the other execution modes. Note also that when performing an import using the command line that the APP execution mode is not valid when connecting directly to the database.

Migration Considerations

In versions of DB.XML before v6.3.3 the default execution mode was SQL. Therefore it is recommended to explicitly select this mode when importing DB.XML files created in earlier versions.

Usage Considerations: APP mode

The valid parent/child relationships are determined by the persistence logic internal to the OTM application server. Therefore, this import mode should normally only be used for data exported using the Migration Entity export type. Content exported via the other methods may still be imported but any custom parent/child relationship tree must be a subset of that produced by the Migration Entity export type.

Transaction Code

The **Transaction Code** specifies how the transaction set is to be processed and will be one of:-

- **I**: Insert new records
- **II**: Insert new record or ignore (i.e. do not fail) if already existing
- **IU**: Insert new records or update if already existing
- **D**: Deletes record.
- **RC**: Replace Children. Delete existing children and replace with new.

Note: The use of the Delete transaction code should be used with care. The delete process attempts to also remove any objects which contain a foreign key to the object being deleted which may not always be the desired effect. Therefore it is strongly advised **NOT** to use this transaction code for object types that may be referenced from transaction data. An example of this is the Location object. Locations can be stand alone but they can also be ship from or ship to locations on Shipments and deleting such locations could have unintended consequences.

Replace Children

When using the **RC** transaction code the child tables that should be involved can be specified as **Managed Tables**. There are also some standard managed tables defined for some data objects which are combined with any managed tables entered as input.

Object Name	**Child Tables**
LOCATION	LOCATION_ACCESSORIAL, LOCATION_ADDRESS, LOCATION_CORPORATION, LOCATION_REFNUM, LOCATION_REMARK, LOCATION_ROLE_PROFILE, LOCATION_SPECIAL_SERVICE, LOCATION_STATUS, LOCATION_ACTIVITY_TIME_DEF
RATE_GEO	RATE_GEO_STOPS, RATE_GEO_ACCESSORIAL, RG_SPECIAL_SERVICE, RG_SPECIAL_SERVICE_ACCESSORIAL, RATE_GEO_COST_GROUP, RATE_GEO_COST, RATE_GEO_COST_WEIGHT_BREAK
RATE_OFFERING	RATE_OFFERING_STOPS, RATE_OFFERING_ACCESSORIAL, RATE_OFFERING_COMMENT
AGENT_EVENT	AGENT_EVENTS_INVALID_ACTION
AGENT	AGENT_EVENT_DETAILS, AGENT_ACTION_DETAILS
CORPORATION	CORPORATION_INVOLVED_PARTY
SAVED_QUERY	SAVED_QUERY_VALUES, SAVED_QUERY_SORT_ORDER
SAVED_CONDITION	SAVED_CONDITION_QUERY
USER_MENU_LAYOUT	USER_MENU_LAYOUT
MONITOR_PROFILE	MONITOR_AGENT, MONITOR_AGENT_LINK

Object Name	**Child Tables**
SHIPMENT	SHIPMENT_STOP, SHIPMENT_STOP_D, SHIPMENT_STOP_REMARK, SHIPMENT_ACCESSORIAL, SHIPMENT_BILL, SHIPMENT_COST, SHIPMENT_COST_REF, SHIPMENT_INVOLVED_PARTY, SHIPMENT_REFNUM, SHIPMENT_REMARK, SHIPMENT_SPECIAL_SERVICE, SHIPMENT_STATUS
STATUS_TYPE	STATUS_VALUE
WORKFLOW_TOPIC_INFO	WORKFLOW_TOPIC_PARAM, WORKFLOW_INFO, WORKFLOW_PARAM
OB_ORDER_BASE	OB_ACCESSORIAL, OB_INVOLVED_PARTY, OB_LINE, OB_LINE_ACCESSORIAL, OB_LINE_ATTRIBUTE, OB_LINE_REFNUM, OB_LINE_REMARK, OB_LINE_SPECIAL_SERVICE, OB_LINE_STATUS, OB_ORDER_BASE, OB_ORDER_BASE_STATUS, OB_REFNUM, OB_REMARK, OB_SHIP_UNIT, OB_SHIP_UNIT_CONTENT, OB_SHIP_UNIT_REFNUM, OB_SHIP_UNIT_REMARK, OB_SHIP_UNIT_SEAL, OB_SHIP_UNIT_STATUS, OB_SPECIAL_SERVICE, OB_SU_ACCESSORIAL, OB_SU_CONTENT_ATTRIBUTE, OB_SU_CONTENT_REFNUM, OB_SU_CONTENT_REMARK, OB_SU_SPECIAL_SERVICE

Refresh Cache

The OTM application maintains a number of in-memory cache objects to improve performance. Historically, DBXML was not able to refresh these cache objects and so occasionally required a restart of the application to pick up some modifications.

When importing using the APP Execution Mode, this new flag is available to indicate that any cache objects associated with the imported data should be refreshed. For example, if the imported data contained a new workflow Agent which is 'active', this agent would automatically subscribe to its listening events and be available to be triggered by, for example, SHIPMENT – MODIFIED events.

Lifetime Events

Whenever certain objects are modified via the application, lifetime events are raised for CREATE, MODIFIED and REMOVED modifications. When the new Lifetime Events flag is used, these events will now also be raised for data imported via DBXML. For example, if a TRANSACTION_SET contains a new LOCATION, the LOCATION – CREATED event will be published.

Commit Scope

By default, each parent element in a TRANSACTION SET is treated as a separate database transaction i.e. if one failed others could potentially succeed.

There is now a new field for Commit Scope which defaults to the current behavior with the scope of ‘PK’. There is a new value of ‘SET’ which indicates that all elements in the TRANSACTION SET must succeed or all will fail.

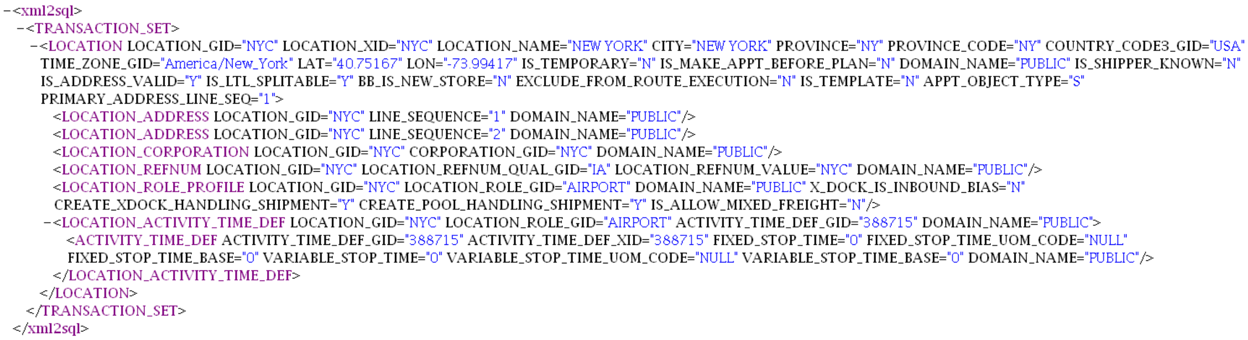
DB.XML User Interface

Exporting DB.XML

This section describes how to export DB.XML using the web-based user interface.

1. Log into Oracle Transportation Management.
2. Locate the DB XML Export user interface. By default this will be Business Process Automation > Data Import/Export > DB.XML Export.
3. Choose an **Export Object Type**. Fields specific to the selected type will now be displayed.
    a. DB Object. The following fields are available:-
        i. DB Object: Object name to be exported. See Using an Object Name.
        ii. Where Clause: For example you can enter DOMAIN_NAME=’GUEST’ or rownum<3. You can also combine the two like this DOMAIN_NAME=’GUEST’ and rownum<3.
    b. DB Object Set. The following fields are available:-
        i. DB Object Set: Object Set name to exported. See Using an Object Set Name.
        ii. Where Clause: For example you can enter DOMAIN_NAME=’GUEST’ or rownum<3. You can also combine the two like this DOMAIN_NAME=’GUEST’ and rownum<3.
    c. Query. The following fields are available:-
        i. SQL Query: For example “select * from activity”.
        ii. Root Name: the element name to be used for the parent XML element.
    d. Migration Entity. The following fields are available:-
        i. Migration Entity Name: For example “Location”
        ii. Object IDs: For example “GUEST.MY_LOC_1, GUEST.MY_LOC_2”.
4. Optionally select to export Foot print columns or Large Objects as elements
5. Click **Run.** Oracle Transportation Management displays the results page.

For example, the following shows an export with DB Object as **LOCATION** and the ‘where clause’ as **LOCATION_GID = ‘NYC’**



```
-<xml2sql>
  -<TRANSACTION_SET>
    -<LOCATION LOCATION_GID="NYC" LOCATION_XID="NYC" LOCATION_NAME="NEW YORK" CITY="NEW YORK" PROVINCE="NY" PROVINCE_CODE="NY" COUNTRY_CODE3_GID="USA"
      TIME_ZONE_GID="America/New_York" LAT="40.75167" LON="-73.99417" IS_TEMPORARY="N" IS_MAKE_APPT_BEFORE_PLAN="N" DOMAIN_NAME="PUBLIC" IS_SHIPPER_KNOWN="N"
      IS_ADDRESS_VALID="Y" IS_LTL_SPLITABLE="Y" BB_IS_NEW_STORE="N" EXCLUDE_FROM_ROUTE_EXECUTION="N" IS_TEMPLATE="N" APPT_OBJECT_TYPE="S"
      PRIMARY_ADDRESS_LINE_SEQ="1">
        <LOCATION_ADDRESS LOCATION_GID="NYC" LINE_SEQUENCE="1" DOMAIN_NAME="PUBLIC"/>
        <LOCATION_ADDRESS LOCATION_GID="NYC" LINE_SEQUENCE="2" DOMAIN_NAME="PUBLIC"/>
        <LOCATION_CORPORATION LOCATION_GID="NYC" CORPORATION_GID="NYC" DOMAIN_NAME="PUBLIC"/>
        <LOCATION_REFNUM LOCATION_GID="NYC" LOCATION_REFNUM_QUAL_GID="IA" LOCATION_REFNUM_VALUE="NYC" DOMAIN_NAME="PUBLIC"/>
        <LOCATION_ROLE_PROFILE LOCATION_GID="NYC" LOCATION_ROLE_GID="AIRPORT" DOMAIN_NAME="PUBLIC" X_DOCK_IS_INBOUND_BIAS="N"
        CREATE_XDOCK_HANDLING_SHIPMENT="Y" CREATE_POOL_HANDLING_SHIPMENT="Y" IS_ALLOW_MIXED_FREIGHT="N"/>
      -<LOCATION_ACTIVITY_TIME_DEF LOCATION_GID="NYC" LOCATION_ROLE_GID="AIRPORT" ACTIVITY_TIME_DEF_GID="388715" DOMAIN_NAME="PUBLIC">
          <ACTIVITY_TIME_DEF ACTIVITY_TIME_DEF_GID="388715" ACTIVITY_TIME_DEF_XID="388715" FIXED_STOP_TIME="0" FIXED_STOP_TIME_UOM_CODE="NULL"
          FIXED_STOP_TIME_BASE="0" VARIABLE_STOP_TIME="0" VARIABLE_STOP_TIME_UOM_CODE="NULL" VARIABLE_STOP_TIME_BASE="0" DOMAIN_NAME="PUBLIC"/>
        </LOCATION_ACTIVITY_TIME_DEF>
      </LOCATION>
    </TRANSACTION_SET>
  </xml2sql>
```

Note: Refer to the Oracle Transportation Management Data Dictionary for more information about what the objects can contain.

Note: Oracle Transportation Management does not display elements that are empty in the database.

Saving DB.XML Output to a File on Your PC

View the source for the frame containing the displayed XML using your browser and save as a file with the ".db.xml" file extension. The steps to view the source vary from browser to browser.

Note: If your output is too large for Notepad, you need to the command line to execute the command.

Note: Especially if your data contains non-ASCII characters, just save your file as-is and use an editor that supports UTF-8 when editing the file later on.

Importing DB.XML

This section describes how to import a DB.XML file using the web-based user interface.

1. Log into Oracle Transportation Management.
2. Locate the DB XML Import user interface. By default this will be Business Process Automation > Data Import/Export > DB.XML Import.
3. Select the appropriate **Schema** and **Execution** mode. These default to GLOGOWNER and APP, respectively.
4. Click **Browse** to specify the required **Input XML File** containing the transaction set to be uploaded**.**
5. The default **Transaction Code** is I (insert). You can change the transactionCode from I to either II, IU, D or RC.
6. If the **Transaction Code** is RC, you may need to specify **Managed Tables**. This will be required when no pre-configured child table properties are setup for the parent table in the input XML file. If no managed tables are found, the RC transaction code is treated like an IU i.e. no child records will be removed.
7. Click **Run**.

Oracle Transportation Management displays summary statistics with a **successCount** and an **errorCount**. The count is the number of transactions that were successful or in error.

DB.XML Command Line Execution

A command line script exists to provide the same capability that is available via the OTM User Interface[1]. The script is called `dbxml.sh or dbxml.bat` (depending on platform) and is present in the `<otm_home>/utils/integration/scripts` directory.

The DB XML command line can operate in two distinct "modes":

- As a remote web client
- As a database client

The choice of mode is determined by the presence of certain parameters. If the "-server" parameter is specified the command will be processed as a remote web client. If "-dbConn" or "-dbURL" are specified the command will be process as a database client.

Some required parameters depend on the mode selected. The following are the required parameters for each mode:

Mode = "remote web client"

Parameter	**Usage**
`server`	The hostname of the web server where the request will be sent via "http". If non-standard ports are used, the format given should be *hostname:port*.
`username`	OTM application user name to be used for execution of database commands. This is required for the correct VPD security.
`password`	Valid password for "username".

Mode = "Database Client"

Parameter	**Usage**

`dbConn`
`or`
`dbURL`
	A connection name configured in the OTM properties e.g. "dbathin". This is the most convenient method as the full JDBC connection URL is constructed by the command processor.
	The JDBC connection URL for the database in the form *jdbc:oracle:thin:@<db server host>:<db listener port>:<svcname>*
	where,
	db server host – hostname of the server where Oracle database is running
	db listener port – port for TNS Listener
	svcname – service name used by OTM DB instance.

---

[1] See OTM Administration Guide for complete instructions on configuring a Java command environment.

Parameter	**Usage**
`dbUser`	Valid database user name to connect to the database. This parameter is optional if the `dbConn` parameter is used as the default user for that connection can be determined from properties files.
`dbPassword`	Valid password for “dbUser”.
`username`	OTM application user name to be used for execution of database commands. This is required for the correct VPD security.
`password`	Valid password for OTM user specified in ‘username’ parameter.

Either the “dbConn” or “dbURL” parameter must be specified.

Exporting DB.XML

Following are the command line parameters for the DBXML export command “xmlExport”:

Parameter	**Usage**
`-dbObjectName`	Specifies the Object Name. See Using an Object Name
`Or`	
`-dbObjectSetName`	Specifies the Object Set Name. See Using an Object Set Name.
`Or`	
`-sqlQuery`	Specifies the SQL Query. See Specifying a SQL Query.
`Or`	
`-sqlFile`	Same as –sqlQuery except that the statement is contained in a file (useful to avoid issues with the command shell and special characters)
`Or`	
`-entityName`	Specifies the Migration Entity. See Migration Entity
`-whereClause`	Used in conjunction with –dbObjectName and –dbObjectSetName to narrow the result set of retrieved objects.
`-rootName`	Used in conjunction with –sqlQuery and –sqlFile to provide the root element of each retrieved object.
`-pkList`	Used in conjunction with –entityName to provide the list of object PKs to be retrieved.
`-footPrint`	Flag available for all commands and indicates if INSERT and UPDATE date/time columns are retrieved.

Parameter	**Usage**
`-useLOBElement`	Flag available for all commands and indicates if LOB column data is to be exported as separate elements or as base 64 encoded attribute values.
`-localDir & -localFileName`	Used to specify that results should be saved to a file rather than displayed to the console.

Using Pre-defined Primary Data Objects

The following is an example of exporting (using remote web server access by default) the first RATE_GEO database object found in the database:

```
dbxml.sh xmlExport -server localhost -username GUEST.ADMIN -password CHANGEME -
dbObjectName RATE_GEO
-whereClause "rownum < 2" -localDir ./ -localFileName rate_geo1.db.xml
```

This example creates the file "rate_geo1.db.xml" in the current working directory.

You need to modify the following arguments specific to your situation:

- Server: hostname of remote web server.
- Username: User name used to login to the remote Oracle Transportation Management instance.
- Password: password corresponding to the username.
- whereClause: SQL 'where' clause used to limit size of export.
- localDir: directory on your PC where output file is written.
- localFileName:(name of local output file. If not specified, defaults to "command.out".

Using a SqlQuery

The following is an example of exporting (using DB client access) all the activity records in the database:

```
dbxml.sh xmlExport -dbConn dbathin -dbUser glogdba -dbPassword password
-username GUEST.ADMIN -password password -sqlQuery "select * from activity"
-rootName ACTIVITY -localDir ./ -localFileName activity.db.xml
```

The above command creates the `activity.db.xml` file in the current working directory.

Importing DB.XML

Following are the command line parameters for the DBXML export command "xmlImport":

Parameter	**Usage**
`-inputXMLFile`	Full Path Name of the file containing the DBXML transaction set to be imported.

Parameter	**Usage**
`-transactionCode`	The transaction code for the import. See the **Reference A: DB.XML Transaction Codes** section for possible transaction codes.
`-exec`	The (optional) mode for the execution. If present, must be one of:- APP (default): persist via application logic. Requires "remote web client" mode. SQL: persist via direct SQL statements to the database PLSQL: persist via PLSQL block execution (Future use: currently delegates to SQL mode). If absent, the default mode is assumed.
`-schema`	If present, must be 'GLOGOWNER' (the default) or 'REPORTOWNER'. If absent, the default is assumed.
`-managedTables`	If the transaction code is RC, this specifies the managed tables that are to be considered for child deletion.
`-updateCache`	Indicator to specify that in memory cache should be refreshed.
`-events`	Indicator to specify that lifetime events should be published for modified data.
`-commitScope`	One of 'PK' (default) or 'SET'. If not present, default is assumed.

You can use `dbxml.sh or dbxml.bat` (depending on platform) to import a client-side db.xml file into a remote Oracle Transportation Management database instance.

Here is a sample command line:

```
dbxml.sh xmlImport -server localhost -username DBA.ADMIN -password CHANGEME -
transactionCode IU -exec APP -inputXMLFile  rate.db.xml
```

Oracle Transportation Management ignores element names that do not correspond to a database table. This allows you to comment your DB.XML file without affecting what is imported.

DB XML Servlet

It may be convenient to export and import DB XML data remotely from the OTM application. This can be achieved in a number of ways; by sending XML messages via HTTP POST to a servlet on the OTM Web Server (discussed in this section) or as a SOAP message to a Web Service on the OTM Application Server (discussed in the next section).

The HTTP POST body should use the format defined below and be sent via HTTP POST to the `glog.integration.servlet.DBXMLServlet.`

The servlet requires authentication using HTTP Basic Authentication. If the network used for communication cannot be assumed to be secure, the HTTPS protocol should be used.

Additionally, the URL **command** parameter should specify which DBXML command should be executed i.e. **xmlImport** for Import and **xmlExport** for Export. A complete example URL would therefore be:

```
http://localhost/GC3/glog.integration.servlet.DBXMLServlet?command=xmlExport
```

Export Message Format

Object Name

The following is the format for the XML message to export XML based on a DB object name:

```
<sql2xml>
   <DBObject>
      <Name>{db object name}</Name>
      <Predicate>{where clause}</Predicate>
   </DBObject>
   <FootPrint>{Y|N}</FootPrint>
   <UseLOBElement>{Y|N}</UseLOBElement>
</sql2xml>
```

For example,

```
<sql2xml>
   <DBObject>
      <Name>LOCATION</Name>
      <Predicate>location_gid = ‘GUEST.MY_LOC’</Predicate>
   </DBObject>
   <FootPrint>N</FootPrint>
   <UseLOBElement>Y</UseLOBElement>
</sql2xml>
```

Object Set

The following is the format for the XML message to export XML based on a DB object set name:

```
<sql2xml>
   <ObjectSet>
      <Name>{db object set name}</Name>
      <Predicate>{where clause}</Predicate>
   </ObjectSet>
   <FootPrint>{Y|N}</FootPrint>
   <UseLOBElement>{Y|N}</UseLOBElement>
</sql2xml>
```

For example,

```
<sql2xml>
   <ObjectSet>
      <Name>DomainReferenceData</Name>
      <Predicate>domain_name = ‘GUEST’</Predicate>
  </ObjectSet>
</sql2xml>
```

Query

The following is the format for the XML message to export XML based on a SQL query:

```
<sql2xml>
   <Query>
      <RootName>{db object name}</RootName>
      <Statement>{where clause}</Statement>
   </Query>
   <FootPrint>{Y|N}</FootPrint>
   <UseLOBElement>{Y|N}</UseLOBElement>
</sql2xml>
```

For example,

```
<sql2xml>
   <Query>
      <RootName>Location</RootName>
      <Statement>SELECT * FROM LOCATION WHERE LOCATION_GID =
‘GUEST.MY_LOC’</Statement>
   </Query>
</sql2xml>
```

Migration entity

The following is the format for the XML message to export based on a Migration Entity name:

```
<sql2xml>
   <Entity>
      <Name>{entity name}</Name>
      <PK>{object PK}</PK>
      …more PK elements…
   </Entity>
   <FootPrint>{Y|N}</FootPrint>
   <UseLOBElement>{Y|N}</UseLOBElement>
</sql2xml>
```

For example,

```
<sql2xml>
   <Entity>
      <Name>Location</Name>
      <PK>GUEST.MY_LOC1</PK>
      <PK>GUEST.MY_LOC2</PK>
   </Entity>
</sql2xml>
```

The response XML will be the TRANSACTION_SET XML identical to that seen in the UI.

Import Message Format

The root element for the DB XML Import message is `xml2sql` and will contain the following:

```
<xml2sql Version=”6.3.3”>
   <TransactionCode>{I|IU|RC}</TransactionCode>
   <SchemaOwner>{schema name}</SchemaOwner>
   <Exec>{APP|SQL|PLSQL}</Exec>
   <UpdateCache>{Y|N}</UpdateCache>
   <RaiseEvents>{Y|N}</RaiseEvents>
   <ManagedTables>
      <Table>{table name 1}</Table>
      <Table>{table name 2}</Table>
   </ManagedTables>
   <TRANSACTION_SET>
       <…table specific elements…>
       …
   </TRANSACTION_SET>
</xml2sql>
```

For example,

```
<xml2sql Version="6.3.3">
   <TransactionCode>I</TransactionCode>
   <SchemaOwner>GLOGOWNER</SchemaOwner>
   <Exec>APP</Exec>
   <UpdateCache>Y</UpdateCache>
   <RaiseEvents>Y</RaiseEvents>
   <TRANSACTION_SET>
       <LOCATION LOCATION_GID='GUEST.MY_LOC'..etc>
          <LOCATION_CORPORATION …etc…/>
          <…etc… other child elements…/>
       </LOCATION>
   </TRANSACTION_SET>
</xml2sql>
```

The response XML will contain the counts for successful or error transactions.

```
<xml2sql>
   <SuccessCount>n</SuccessCount>
   <ErrorCount>m</ErrorCount>
   <ElapsedTime>p</ElapsedTime>
   <TimePerTransaction>q</TimePerTransaction>
</xml2sql>
```

Where `n, m, p` & `q` are integers and `p` & `q` are the number of milliseconds.

DB XML Web Service

DB XML Export and Import can also be performed by calling a SOAP Web Service running on the Application server. The WSDL for the service will be located under:

```
http://<server:port>/GC3Services/glog.integration.webservice.command.CommandService?WSDL
```

Where server and port are specific to the host and port configured for the WebLogic server running OTM.

The service is secured via Web Service Security in common with all other OTM Web Services and so by default requires the WSS Username Token Profile over HTTPS for authentication.

The SOAP messages (defined in the WSDL) are essentially identical to the messages used for the DB XML Servlet but will be 'wrapped' in the corresponding command/operation name, i.e.:

```
<xmlExport>
   <sql2xml>
     <…elements as described previously>
   </sql2xml>
</xmlExport>
```

and

```
<xmlImport>
   <xml2sql>
     <…elements as described previously>
   </xml2sql>
</xmlImport>
```

Editing DB.XML Files

This section describes how you edit an exported DB.XML file before importing it again.

A Sample DB.XML File

An exported DB.XML file might look like this. Note that the content is wrapped in a pair of <TRANSACTION_SET> tags.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xml2sql Version=”6.3.3”>
   <TRANSACTION_SET>
   <CORPORATION CORPORATION_GID="ACL" CORPORATION_XID="ACL"
   DOMAIN_NAME="PUBLIC" INSERT_DATE="2001-10-05 19:03:37"
   INSERT_USER="DBA.ADMIN" IS_DOMAIN_MASTER="N" UPDATE_DATE="2001-10-06
   12:43:46" UPDATE_USER="DBA.GLOGLOAD" dbObjectName="CORPORATION" />
</TRANSACTION_SET>
</xml2sql>
```

You can edit the values and add new objects.

When editing date and time values, be sure to keep the following format: YYYY-MM-DD HH:MM:SS.

If you miss an element in the exported file this is probably because Oracle Transportation Management does not export elements that are empty in the database. This means that you will have to add the tag to the DB.XML file yourself. Refer to the Oracle Transportation Management Data Dictionary for more information about what objects and tables exist.

Oracle Transportation Management ignores element names that do not correspond to the database table. This allows you to comment your DB.XML file without affecting what is imported.

As you edit the file, keep all element and attribute names in uppercase.

3. Loading CSV Data via the Command Line

This chapter describes how to import and export CSV from the command line.

Importing and Exporting on the Server Side

This section describes how to use CSVUtil to export and import data from a local Oracle Transportation Management database.

CSVUtil has the following syntax and arguments.

```
java glog.database.admin.CSVUtil –command
<i|ii|iu|u|uu|d|dd|xcsv|xcsvcd|xcsvpcd|xcsvpd|xsql> -connectionId
<connectionId> -tableName <tableName> -dataDir <dataDirectory> -dataFileName
<dataFileName> -appendFile –runsqlloader -domain_name <domainName> -useT2 <Y|N>
-debug -XMLCSVOutput -sqlQuery <queryString> -whereClause <whereClause> -
clobDir <clobDirectory> –xvalidate <Y|N> -encoding <encoding>
```

CSVUtil supports the following commands and arguments:

Commands	**Arguments**
command	i - insert CSV data into the database ii - insert data, while suppressing unique key constraint violations iu - attempts to insert data. If a primary key violation occurs, it updates the data. No delete statements are generated. u - update data in the database uu - update data, while suppressing “no data found” constraint violations d- delete data from the database dd- delete data, while suppressing “no data found” constraint violations xcsv - export a CSV file xcsvcd - export a multi-table CSV file with all subordinate child tables (e.g. shipment_stop, shipment_stop_d etc. for the shipment table). A table set called C.<table_name> controls which tables are considered to be children of a given table. For example, the C.SHIPMENT table set contains the following tables: shipment_stop, shipment_refnum, shipment_remark, etc. Similarly, the C.SHIPMENT_STOP table_set contains the shipment_stop_d table. If you log in as DBA.ADMIN in Oracle Transportation Management, you can use the Table Set Manager to modify the contents of the various C.* table sets. xcsvpcd - export a multi-table CSV file with both parent and child data. xcsvpd - export a multi-table CSV file with all referenced non-public foreign key records (parent data) required to load the record(s) in a foreign database. xsql - export data as a script of insert statements rather than a CSV file

Commands	**Arguments**

connectionId	The connectionId is a shorthand method for providing an Oracle username, password, and server.
	For example, if you specify your connectionId as codegen, you need to add the following properties to your glog.properties file:
	glog.database.codegen.schema=glogowner
	glog.database.codegen.t2client.driverClassName=oracle.jdbc.driver.OracleDriver
	glog.database.codegen.t2client.databaseURL=jdbc:oracle:thin:@localhost:1521
	glog.database.codegen.user=glogload
	glog.database.codegen.password=glogload
	glog.database.codegen.server=dbserver
	glog.database.codegen.t2client.pool=

tableName	The tableName argument is only specified for the xcsv and xsql commands. This specifies the name of the database table to export. Can be null if sqlQuery is specified. Must be upper case.
dataDir	The dataDir argument specifies the location to either read or write the file specified in the –dataFileName argument. The following glog.property file setting controls the default value of the dataDir argument: glog.database.load.dir=d:\\upload In this case, the default directory has been set to d:\upload. Note that two backslashes are required in glog.properties.
dataFileName	The dataFileName argument specifies the name of the file in the dataDir directory to either read or write. This field is required when importing a file, but is optional when exporting a file. If unspecified for an export, the output is written to System.out.
appendFile	The appendFile argument only applies to the export commands (xcsv and xsql). If specified, CSVUtil will append to the file specified by the dataFileName argument instead of overwriting it.

Commands	**Arguments**
removeUndefinedColumns	CSVUtil supports, by default, the ability to ignore columns that are not defined in the target table. This is especially useful when exporting from a migrated database with deprecated columns, into a newly created database that does not have the deprecated columns. There is some performance impact for this feature. To deactivate the feature, use the following command line option: -removeUndefinedColumns N This option is only available when running CSVUtil directly on the command line. It is not available using either the web or ClientUtil.
runsqlloader	The runsqlloader argument only applies to import commands. If specified, the oracle sqlloader program will be used to load the CSV file instead of a java procedure. If you have sqlloader installed on your system the sqlloader is faster than the java procedure.
-maxError	By default in CSVUtil, after 50 errors occur, processing stops. You can change this default value to make it higher or lower using the –maxError command line argument. For example: -maxError 20 This parameter is currently only available when running CSVUtil as a java application on the command line.
domain_name	The domain_name argument only applies to the export commands (xcsv and xsql). It specifies that only the data in that domain is to be exported.
useT2	Used to avoid using the T2Connection class, which depends on VPD being already setup correctly. When loading certain Oracle Transportation Management "system" tables, it is necessary to avoid the use of the T2 connection class (it’s a chicken or the egg type situation). For normal data loading, using the T2Connection class is correct and desirable.
debug	Used for debugging.
XMLCSVOutput	If true, then output looks like this: <TableName></TableName> or <SqlQuery>...</SqlQuery> <ColumnList></ColumnList> <ExecSQL></ExecSQL> <Row>...</Row> <Row>...</Row>
sqlQuery	If specified, then xcsv command is required and tableName is ignored.

Commands	**Arguments**
whereClause	Only used when tableName is specified and domainName is omitted.
clobDir	Directory where external CLob files are read. Only used when importing external CLob files and not using sqlloader.
xvalidate	Can be either Y (default) or N. When set to Y, CSVUtil gives you more user-friendly diagnostics messages and hinders missing values in your CSV file to delete an existing value in the database. If you want CSVUtil to allow data to be nulled out, you should specify xvalidate as N when running CSVUtil.
encoding	The encoding of the file you import. Common settings are ISO-8859-1 (default) and UTF-8. You especially need to consider this when you import data containing characters outside the 7-bit ASCII set. Also, consider the encoding of your database.

CLobs in CSV Files

CSVUtil supports inserting, updating, and deleting CLobs. You can:

- Include the CLob in the CSV file (each CLob<1Mb, no newline characters)
- In the CSV file, refer to an external file holding the CLob. (no size restrictions on the CLobs, newline characters allowed)

  Note: CSVUtil can only handle one CLob per record.

Here is a sample CSV file that inserts a CLob using the in-line method:

```
CLOB_TEST
SEQ,DESCR,XML
9,"THIS IS SO COOL",<asdf>blahblah</asdf>
10,"LINE2",<qwerty>yaya</qwerty>
```

In this case, the "XML" column is of type CLob. When using the in-line method, each CLob:

- Must be specified on a single line (no newline characters).
- Must be smaller than 1 megabyte.

Here is a sample CSV file that inserts two CLobs using the external file method:

```
CLOB_TEST
SEQ,DESCR,EXT_FNAME,XML
11,"THIS IS SO COOL",myxmlfile.xml
12,"LINE2",myxmlfile2.xml
```

When using the external file method, you must specify a special "pseudo column" called "EXT_FNAME ". The EXT_FNAME pseudo column must be specified to the left of the CLob column. In this case, you will have an extra column on line 2. So in this case, line 2 has 4 columns, but there are only 3 columns in the data lines.

The external file method must be used when inserting CLobs containing newline characters, or when inserting CLobs greater than 1 megabyte.

Exporting With Parent Data

To export a data record with its parent data, you can do the following:

```
java glog.database.admin.CSVUtil -command xcsvwpd -tableName SHIPMENT -
whereClause "shipment_gid = 'MDIETL.184'" -connectionId angel37
```

The above command exports the record for shipment MDIETL.184, along with all the referenced non-public foreign key records required to successfully load the SHIPMENT record in a foreign database. The generated CSV file is in multi-table format.

> **Note:** All the xcsvw* commands are far more expensive in terms of CPU usage than the plain xcsv command. Using them to export a large data set will take a long time, since many foreign keys must be found. Use the commands with a restrictive where-clause, as shown in the examples, to limit the running time.

Exporting With Child Data

To export a data record with its child data, you can do the following:

```
java glog.database.admin.CSVUtil -command xcsvwcd -tableName SHIPMENT -
whereClause "shipment_gid = 'MDIETL.184'" -connectionId angel37
```

The above command exports the record for shipment MDIETL.184, along with all the subordinate child tables such as shipment_stop, shipment_stop_d etc.

Exporting With Both Parent and Child Data

To export a data record with both its parent and child data, you can do the following:

```
java glog.database.admin.CSVUtil -command xcsvwpcd -tableName SHIPMENT -
whereClause "shipment_gid = 'MDIETL.184'" -connectionId angel37
```

This command should be used with care since it can take while to run.

GL_User Table

CSVUtil supports adding and deleting records in the GL_USER table. This table stores the Oracle Transportation Management users and their passwords.

When the GL_USER table is specified in the header of a CSV file, special processing is done.

If you are an authorized GL_USER, you may add and delete records in the GL_USER table. As an exception for this table, you can only use the commands: i, ii, d, or dd.

> **Note:** The u, uu, and iu commands are not supported when loading the GL_USER table.

4. Loading CSV Data via Web Pages

Running CSVUtil via the command line is only possible if your client environment is configured correctly. If your client environment is not configured, you can still run CSVUtil via the web.

Importing

This section describes how to import a CSV file using Oracle Transportation Management.

1. Log in to Oracle Transportation Management.
2. Choose Business Process Automation > Integration > Integration Manager.
3. Click Upload an XML / CSV Transmission.
4. Select the file to upload. The upload will transfer files from your local machine to the server.

   Note: You must select a .CSV file.

5. Click **Upload** and Oracle Transportation Management displays the page for importing the file. If you select a file other than a .CSV file, a different page will open.
6. If it is not already selected, select *i* from the **command** list**.**
7. Leave the **dataDir** as is.
8. Leave the **dataFileName** as is.
9. If you are loading a large file, you may specify the **runsqlloader** option. This will only work if sqlloader is installed on the Oracle Transportation Management web server. The following line must be added to the jserv.properties file to make sqlloader run from the web:

   ```
   wrapper.path = d:/product/oracle/ora81/bin
   ```

   This entry would be different depending on the location of the Oracle bin directory.

10. The **xvalidate** drop-down list allows you to turn off verbose diagnostic messaging. To leave messaging on, the value in the drop down list should be Y, which is the default.
11. In the encoding drop down list, select the appropriate encoding type for your CSV file. If your file contains standard ASCII characters, then it can be encoded as ISO-8859-1. If it contains non-standard, international characters, then it should be encoded as UTF-8.
12. Click **Run** and Oracle Transportation Management displays a results page.

To read more about interpreting error messages, see the **Reference C: CSVUtil Response Messages** section.

5. Loading Rate Data via CSV

This chapter gives you examples of:

- The tables you need to import to set up rates in Oracle Transportation Management.
- How to format the CSV files.
- The order in which you must import tables.

Refer to the Oracle Transportation Management Data Dictionary to learn what data you need and in what order you need to import it.

> **Note:** Any blank columns are not included in the CSV files. See the Data Dictionary for a complete list of columns.

Importing Location Information

This section describes how to import location information in CSV format. A set of sample CSV files is presented. Tables must be loaded in the order presented in this section. Otherwise, foreign key violations occur.

1. Import the LOCATION Table.

   The following example illustrates how you specify LOCATION data in CSV format.

   ```
   LOCATION
   LOCATION_GID,LOCATION_XID,LOCATION_NAME,CITY,PROVINCE,PROVINCE_CODE,POSTAL_C
   ODE,COUNTRY_CODE3_GID,TIME_ZONE_GID,LAT,LON,IS_TEMPORARY,IS_MAKE_APPT_BEFORE
   _PLAN,RATE_CLASSIFICATION_GID,DOMAIN_NAME,IS_SHIPPER_KNOWN,IS_ADDRESS_VALID,
   IS_MIXED_FREIGHT_THU_ALLOWED,SLOT_TIME_INTERVAL,SLOT_TIME_INTERVAL_UOM_CODE,
   SLOT_TIME_INTERVAL_BASE,IS_LTL_SPLITABLE
   EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
   MYDOMAIN.YELLOW,YELLOW,YELLOW
   LOCATION,PITTSBURGH,,PA,99999,USA,America/New_York,,,N,N,,MYDOMAIN,N,U,N,,,,
   Y
   MYDOMAIN.MYLOCATION,MYLOCATION,MYLOCATION,PHILADELPHIA,,PA,19001,USA,America
   /New_York,40.12726,-75.12881,N,N,COMMERCIAL,MYDOMAIN,N,U,N,0,S,0,Y
   MYDOMAIN.MYCORPORATION,MYCORPORATION,MYCORPORATION,PHILADELPHIA,,PA,19001,US
   A,America/New_York,40.12726,-75.12881,N,N,COMMERCIAL,MYDOMAIN,N,U,N,0,S,0,Y
   ```

2. Import the LOCATION_ADDRESS table

   The following example illustrates how you specify LOCATION_ADDRESS data in CSV format.

   ```
   LOCATION_ADDRESS
   LOCATION_GID,LINE_SEQUENCE,ADDRESS_LINE,DOMAIN_NAME
   EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
   MYDOMAIN.YELLOW,1,432 YELLOW AVE,MYDOMAIN
   MYDOMAIN.MYCORPORATION,1,11 EMPEROR AVE,MYDOMAIN
   MYDOMAIN.MYLOCATION,1,123 MAPLE STREET,MYDOMAIN
   MYDOMAIN.MYLOCATION,2,BUILDING H,MYDOMAIN
   MYDOMAIN.MYLOCATION,3,ROOM 100,MYDOMAIN
   ```

3. Import the CORPORATION Table.

   The following example illustrates how you specify CORPORATION data in CSV format.

Note: Each CORPORATION_GID must correspond to a LOCATION_GID specified in the location table (See example).

```
CORPORATION
CORPORATION_GID,CORPORATION_XID,CORPORATION_NAME,DOMAIN_NAME,IS_DOMAIN_MASTE
R,IS_SHIPPING_AGENTS_ACTIVE,IS_ALLOW_HOUSE_COLLECT
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.MYCORPORATION,MYCORPORATION,MYCORP,MYDOMAIN,N,N,N
MYDOMAIN.YELLOW INC,YELLOW INC,YELLOW INCORPORATED,MYDOMAIN,N,N,N
```

4. Import the LOCATION_CORPORATION Table.

   The following example illustrates how you specify LOCATION_CORPORATION data in CSV format. This links a location to a corporation.

```
LOCATION_CORPORATION
LOCATION_GID,CORPORATION_GID,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.MYLOCATION,MYDOMAIN.MYCORPORATION,MYDOMAIN
MYDOMAIN.MYCORPORATION,MYDOMAIN.MYCORPORATION,MYDOMAIN
MYDOMAIN.YELLOW,MYDOMAIN.YELLOW INC,MYDOMAIN
```

5. Import the SERVPROV Table.

   The following example illustrates how you specify SERVPROV data in CSV format. Each SERVPROV_GID must correspond to a LOCATION_GID specified in the location table (See example).

```
SERVPROV
SERVPROV_GID,SERVPROV_XID,AUTO_PAYMENT_FLAG,DOMAIN_NAME,IS_DISPATCH_BY_REGIO
N,ALLOW_TENDER,IS_ACCEPT_SPOT_BIDS,IS_ACCEPT_BROADCAST_TENDERS,IS_LOCALIZE_B
ROADCAST_CONTACT,DO_CONDITIONAL_ACCEPTS,IS_INTERNAL_NVOCC,IS_ACCEPT_BY_SSU_A
LLOWED,IS_COPY_INV_DELTA_BACK_TO_SHIP,INVOICING_PROCESS
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.YELLOW,YELLOW,N,MYDOMAIN,N,Y,N,N,N,N,N,N,N,S
```

6. Import the LOCATION_ROLE_PROFILE Table.

   The following example illustrates how you specify LOCATION_ROLE_PROFILE data in CSV format. Each location should have at least one row in this table.

```
LOCATION_ROLE_PROFILE
LOCATION_GID,LOCATION_ROLE_GID,CALENDAR_GID,FIXED_HANDLING_TIME,FIXED_HANDLI
NG_TIME_UOM_CODE,FIXED_HANDLING_TIME_BASE,CREATE_XDOCK_HANDLING_SHIPMENT,CRE
ATE_POOL_HANDLING_SHIPMENT,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.YELLOW,CARRIER,,0,S,0,N,N,MYDOMAIN
MYDOMAIN.MYLOCATION,SHIPFROM/SHIPTO,,0,S,0,N,N,MYDOMAIN
MYDOMAIN.MYCORPORATION,BILL TO,,0,S,0,N,N,MYDOMAIN
MYDOMAIN.MYCORPORATION,REMIT TO,,0,S,0,N,N,MYDOMAIN
```

7. Import the LOCATION_REMARK Table.

   The following example illustrates how you specify LOCATION_REMARK data in CSV format.

```
LOCATION_REMARK
LOCATION_GID,REMARK_SEQUENCE,REMARK_QUAL_GID,REMARK_TEXT,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
```

```
MYDOMAIN.MYLOCATION,1,REM,DRIVER CANNOT HAVE A BEARD,MYDOMAIN
MYDOMAIN.MYLOCATION,2,REM,DRIVER MUST HAVE SAFETY GLASSES,MYDOMAIN
```

Importing Service Times

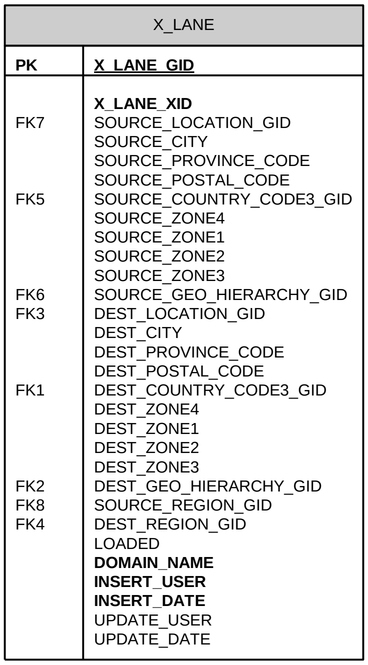
The following example illustrates how you specify SERVICE_TIME data in CSV format.

```
SERVICE_TIME
X_LANE_GID,RATE_SERVICE_GID,SERVICE_TIME_VALUE,SERVICE_DAYS,DOMAIN_NAME,SERVICE
_TIME_VALUE_UOM_CODE,SERVICE_TIME_VALUE_BASE
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.194-064,MYDOMAIN.VOYAGE-DEFAULT,172800,,MYDOMAIN,S,172800
MYDOMAIN.194-065,MYDOMAIN.VOYAGE-DEFAULT,86400,,MYDOMAIN,S,86400
```

In the above example, note that you must specify SERVICE_DAYS, and leave the SERVICE_TIME_VALUE unspecified. As an alternative, you must specify SERVICE_TIME_VALUE in seconds, and leave the SERVICE_DAYS unspecified. You must never specify both a SERVICE_TIME_VALUE and a SERVICE_DAYS value on the same record.

Importing X_LANE Data for Rates

This section provides an example for loading X_LANE data in CSV format. Typically, the X_LANE tables are loaded prior to the loading of the RATE_GEO and RATE_GEO_COST tables.



X_LANE	
PK	**X_LANE_GID**
	X_LANE_XID
FK7	SOURCE_LOCATION_GID
	SOURCE_CITY
	SOURCE_PROVINCE_CODE
	SOURCE_POSTAL_CODE
FK5	SOURCE_COUNTRY_CODE3_GID
	SOURCE_ZONE4
	SOURCE_ZONE1
	SOURCE_ZONE2
	SOURCE_ZONE3
FK6	SOURCE_GEO_HIERARCHY_GID
FK3	DEST_LOCATION_GID
	DEST_CITY
	DEST_PROVINCE_CODE
	DEST_POSTAL_CODE
FK1	DEST_COUNTRY_CODE3_GID
	DEST_ZONE4
	DEST_ZONE1
	DEST_ZONE2
	DEST_ZONE3
FK2	DEST_GEO_HIERARCHY_GID
FK8	SOURCE_REGION_GID
FK4	DEST_REGION_GID
	LOADED
	DOMAIN_NAME
	INSERT_USER
	INSERT_DATE
	UPDATE_USER
	UPDATE_DATE

The following example illustrates how you specify GEO_HIERARCHY and X_LANE data in CSV format.

```
GEO_HIERARCHY
GEO_HIERARCHY_GID,GEO_HIERARCHY_XID,RANK,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.USZIP4,USZIP4,4,MYDOMAIN

X_LANE
```

```
X_LANE_GID,X_LANE_XID,SOURCE_POSTAL_CODE,SOURCE_COUNTRY_CODE3_GID,SOURCE_GEO_HI
ERARCHY_GID,DEST_POSTAL_CODE,DEST_COUNTRY_CODE3_GID,DEST_GEO_HIERARCHY_GID,DOMA
IN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.194-064,194-
064,194,USA,MYDOMAIN.USZIP4,64,USA,MYDOMAIN.USZIP4,MYDOMAIN
MYDOMAIN.194-065,194-
065,194,USA,MYDOMAIN.USZIP4,65,USA,MYDOMAIN.USZIP4,MYDOMAIN
MYDOMAIN.MY LANE,MY LANE,194,,POSTAL_CODE,64,,POSTAL_CODE,MYDOMAIN
```

Importing LTL Rates

This section describes how to specify LTL rates and gives sample CSV files for several scenarios.

The following tables must be loaded (in order):

- RATE_OFFERING (setup manually on Oracle Transportation Management web pages)
- X_LANE (see the **Importing X_LANE Data for Rates** section.)
- RATE_GEO
- ACCESSORIAL_CODE
- ACCESSORIAL_COST
- RATE_GEO_ACCESSORIAL $^{(*)}$
- RATE_GEO_COST_GROUP
- RATE_GEO_COST
- RATE_UNIT_BREAK_PROFILE
- RATE_UNIT_BREAK
- RATE_GEO_COST_UNIT_BREAK

> **Note:** (*) RATE_GEO_ ACCESSORIAL must come after RATE_GEO, but is not required before the remaining tables.

Assumptions:

- You have loaded the rate offering table using Oracle Transportation Management web pages
- You have loaded the X_Lane table (see the **Importing X_LANE Data for Rates** section.)

Simplified ERD for LTL Rates

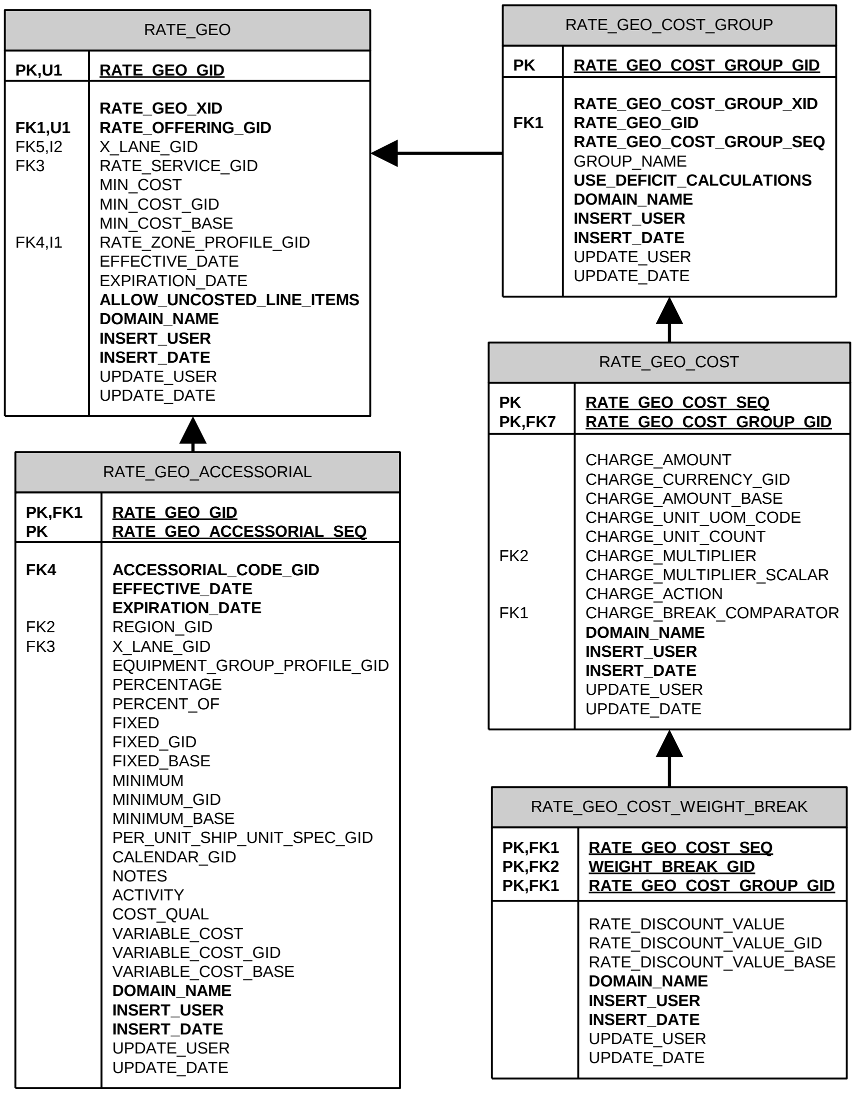


Table Notes:

- RATE_GEO Table

Allow_uncosted_line_items in Y/N (defaults to "N")

- RATE_GEO_ACCESSORIAL

Left_Operand1 – Basis options define what variable you want to base your conditional charge on.

Oper1_gid – The operand you compare with.

Low_value1 – Depending on the operand you use, you might need only the low_value1 or additionally the high_value1.

- RATE_GEO_COST_GROUP Table

Use_deficit_calculations in Y/N (defaults to "N")

- RATE_GEO_COST Table

charge_unit_uom_code - unit of measure (e.g. “LB” for pounds, or “MI” for miles)

charge_unit_count - hundredweight, etc.

charge_action – add (A), setmin (M), setmax (X), multiply/discount (D)

charge_break_comparator -identifies data element used to access the break

Scenario–Based on Simple Unit Breaks

This scenario assumes that rates are defined as simple unit breaks.

1. Import RATE_GEO table.

```
RATE_GEO
RATE_GEO_GID,RATE_GEO_XID,RATE_OFFERING_GID,MIN_COST,MIN_COST_GID,MIN_COST_B
ASE,X_LANE_GID,DOMAIN_NAME
"MYDOMAIN.194-064","194-064","MYDOMAIN.YELLOW",1.0,"USD",1.0,"MYDOMAIN.194-
064","MYDOMAIN"
"MYDOMAIN.194-065","194-065","MYDOMAIN.YELLOW",1.0,"USD",1.0,"MYDOMAIN.194-
065","MYDOMAIN"
```

2. Import RATE_GEO_COST_GROUP table.

```
RATE_GEO_COST_GROUP
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_GROUP_XID,RATE_GEO_GID,RATE_GEO_COST_G
ROUP_SEQ,GROUP_NAME,DOMAIN_NAME
"MYDOMAIN.194-064","194-064","MYDOMAIN.194-064",1,"MY_GROUP_NAME","MYDOMAIN"
"MYDOMAIN.194-065","194-065","MYDOMAIN.194-065",1,"MY_GROUP_NAME","MYDOMAIN"
```

3. Import RATE_GEO_COST table.

```
RATE_GEO_COST
RATE_GEO_COST_SEQ,RATE_GEO_COST_GROUP_GID,CHARGE_UNIT_UOM_CODE,CHARGE_UNIT_C
OUNT,CHARGE_BREAK_COMPARATOR,DOMAIN_NAME
1,"MYDOMAIN.194-064","LB",100,"SHIPMENT.WEIGHT","MYDOMAIN"
1,"MYDOMAIN.194-065","LB",100,"SHIPMENT.WEIGHT","MYDOMAIN"
```

4. Import RATE_UNIT_BREAK_PROFILE table.

```
RATE_UNIT_BREAK_PROFILE
RATE_UNIT_BREAK_PROFILE_GID,RATE_UNIT_BREAK_PROFILE_XID,DATA_TYPE,LOOKUP_TYP
E,UOM_TYPE,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.LT 1000,LT 1000,U,M,WEIGHT,MYDOMAIN
```

5. Import RATE_UNIT_BREAK table.

```
RATE_UNIT_BREAK
RATE_UNIT_BREAK_GID,RATE_UNIT_BREAK_XID,RATE_UNIT_BREAK_PROFILE_GID,RATE_UNI
T_BREAK_MAX,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.1000,0-1000,MYDOMAIN.LT 1000,1000 LB,MYDOMAIN
MYDOMAIN.1000-3000,1000-3000,MYDOMAIN.LT 1000,3000 LB,MYDOMAIN
```

6. Import RATE_GEO_COST_UNIT_BREAK table.

```
RATE_GEO_COST_UNIT_BREAK
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_SEQ,RATE_UNIT_BREAK_GID,CHARGE_AMOUNT,
CHARGE_AMOUNT_GID,CHARGE_AMOUNT_BASE,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.194-064,1,MYDOMAIN.1000,48.53,USD,48.53,MYDOMAIN
MYDOMAIN.194-064,1,MYDOMAIN.1000-3000,37.56,USD,37.56,MYDOMAIN
```

Scenario–Based on Cost Per Pound, Surcharge, and Discount

This scenario assumes that:

- Freight cost is $0.07 per lb
- Fuel Surcharge is 3% of Total Cost (Accessorial)
- Discount is 65% of Total Cost
- There is a $50 allowance for loading
- The minimum charge is based on 10,000 lb
- Total Cost = (weight * 0.07 – 50.00) * (65% Discount) * (Accessorial Surcharge of 3%)
- Min Cost = (10,000 * 0.07 – 50.00) * (1 - 0.65) * (1.03) = 234.325
-

Summary

1. Import RATE_GEO table.

```
RATE_GEO
RATE_GEO_GID,RATE_GEO_XID,RATE_OFFERING_GID,MIN_COST,MIN_COST_GID,MIN_COST_B
ASE,X_LANE_GID,DOMAIN_NAME
"MYDOMAIN.194-064-2","194-064-
2","MYDOMAIN.YELLOW",234.325,"USD",234.325,"MYDOMAIN.194-064","MYDOMAIN"
```

2. Import ACCESSORIAL_COST table.

```
ACCESSORIAL_COST
ACCESSORIAL_COST_GID,ACCESSORIAL_COST_XID,CHARGE_MULTIPLIER,CHARGE_MULTIPLIE
R_SCALAR,CHARGE_ACTION,CHARGE_TYPE,USE_DEFAULTS,CHARGE_MULTIPLIER_OPTION,USE
S_UNIT_BREAKS,DOMAIN_NAME,IS_FILED_AS_TARIFF
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FS,FS,SHIPMENT.COSTS.AMOUNT,1.03,A,B,N,A,N,MYDOMAIN,N
```

3. Import ACCESSORIAL_CODE table.

```
ACCESSORIAL_CODE
ACCESSORIAL_CODE_GID,ACCESSORIAL_CODE_XID,ACCESSORIAL_DESC,APPLY_GLOBALLY,DO
MAIN_NAME,IS_FLOW_THRU,IS_VAT_EXEMPT
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FUEL_SURCHARGE,FUEL_SURCHARGE,FUEL SURCHARGE,Y,MYDOMAIN,N,N
```

4. Import RATE_GEO_ACCESSORIAL table.

```
RATE_GEO_ACCESSORIAL
ACCESSORIAL_COST_GID,RATE_GEO_GID,ACCESSORIAL_CODE_GID,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FS,MYDOMAIN.194-064-2,MYDOMAIN.FUEL_SURCHARGE,MYDOMAIN
```

5. Import RATE_GEO_COST_GROUP table.

```
RATE_GEO_COST_GROUP
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_GROUP_XID,RATE_GEO_GID,RATE_GEO_COST_G
ROUP_SEQ,GROUP_NAME,DOMAIN_NAME
MYDOMAIN.194-064-2,194-064-2,MYDOMAIN.194-064-2,1,MY_GROUP_NAME_2,MYDOMAIN
```

6. Import RATE_GEO_COST table.

```
RATE_GEO_COST
RATE_GEO_COST_SEQ,RATE_GEO_COST_GROUP_GID,CHARGE_AMOUNT,CHARGE_CURRENCY_GID,
CHARGE_AMOUNT_BASE,CHARGE_UNIT_UOM_CODE,CHARGE_UNIT_COUNT,CHARGE_MULTIPLIER,
CHARGE_MULTIPLIER_SCALAR,CHARGE_ACTION,DOMAIN_NAME
1,"MYDOMAIN.194-064-
2",0.07,"USD",0.07,"LB",1,"SHIPMENT.WEIGHT",,"A","MYDOMAIN"
2,"MYDOMAIN.194-064-2",-50.0,"USD",-50.0,,1,,,"A","MYDOMAIN"
3,"MYDOMAIN.194-064-2",,,,,1,,0.35,"D","MYDOMAIN"
```

Note: An alternative to using the data specified for the RATE_GEO_ACCESSORIAL table above would be to add another Sequence to this table with the following (representing a 3% surcharge of the total value):

```
4,"MYDOMAIN.194-064-2",,,,,1,,1.03,"D","MYDOMAIN"
```

Scenario–Based on Cost Per Pound, Conditional Surcharge, Global Surcharge, and Discount

This scenario assumes that:

- Freight cost is $0.07 per lb
- Unload fee is $10 if the weight > 20000lb (Accessorial)
- Fuel Surcharge is 3% of Total Cost (Accessorial)
- Discount is 65% of Total Cost
- There is a $50 allowance for loading
- The minimum charge is based on 10,000 lb

Summary

- Total Cost = ((weight * 0.07 – 50.00) * (65% Discount) + (if weight>20000lb then Accessorial Surcharge of 10)) * (1.03)
- Min Cost = (10,000 * 0.07 – 50.00) * (1 - 0.65) * (1.03) = 234.325

1. Import RATE_GEO table.

```
RATE_GEO
RATE_GEO_GID,RATE_GEO_XID,RATE_OFFERING_GID,MIN_COST,MIN_COST_GID,MIN_COST_B
ASE,X_LANE_GID,DOMAIN_NAME
MYDOMAIN.194-064-3,194-064-
3,MYDOMAIN.YELLOW,234.325,USD,234.325,MYDOMAIN.194-064,MYDOMAIN
```

2. Import ACCESSORIAL_COST table.

```
ACCESSORIAL_COST
```

```
ACCESSORIAL_COST_GID,ACCESSORIAL_COST_XID,LEFT_OPERAND1,OPER1_GID,LOW_VALUE1
,AND_OR1,LEFT_OPERAND2,OPER2_GID,LOW_VALUE2,CHARGE_MULTIPLIER,CHARGE_AMOUNT,
CHARGE_AMOUNT_GID,CHARGE_AMOUNT_BASE,CHARGE_UNIT_COUNT,CHARGE_MULTIPLIER_SCA
LAR,CHARGE_ACTION,CHARGE_TYPE,USE_DEFAULTS,CHARGE_MULTIPLIER_OPTION,USES_UNI
T_BREAKS,DOMAIN_NAME,ROUNDING_TYPE,ROUNDING_FIELDS_LEVEL,ROUNDING_APPLICATIO
N,IS_FILED_AS_TARIFF
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FS,FS,,,,,,,,SHIPMENT.COSTS.AMOUNT,,,,,1.03,A,B,N,A,N,MYDOMAIN,N,0,
A,N
MYDOMAIN.FS-2,FS-
2,SHIPMENT.STOPS.SHIPUNITS.ACTIVITY,EQ,D,S,SHIPMENT.STOPS.WEIGHT,GT,20000
LB,SHIPMENT,10,USD,10,1,,A,B,N,A,N,MYDOMAIN,,,,N
```

3. Import ACCESSORIAL_CODE table.

```
ACCESSORIAL_CODE
ACCESSORIAL_CODE_GID,ACCESSORIAL_CODE_XID,ACCESSORIAL_DESC,APPLY_GLOBALLY,DO
MAIN_NAME,IS_FLOW_THRU,IS_VAT_EXEMPT
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FUEL_SURCHARGE,FUEL_SURCHARGE,FUEL SURCHARGE,Y,MYDOMAIN,N,N
```

4. Import RATE_GEO_ACCESSORIAL table.

```
RATE_GEO_ACCESSORIAL
ACCESSORIAL_COST_GID,RATE_GEO_GID,ACCESSORIAL_CODE_GID,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FS-2,MYDOMAIN.194-064-3,MYDOMAIN.FUEL_SURCHARGE,MYDOMAIN
MYDOMAIN.FS,MYDOMAIN.194-064-3,MYDOMAIN.FUEL_SURCHARGE,MYDOMAIN
```

5. Import RATE_GEO_COST_GROUP table.

```
RATE_GEO_COST_GROUP
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_GROUP_XID,RATE_GEO_GID,RATE_GEO_COST_G
ROUP_SEQ,GROUP_NAME,DOMAIN_NAME
MYDOMAIN.194-064-3,194-064-3,MYDOMAIN.194-064-3,1,MY_GROUP_NAME_3,MYDOMAIN
```

6. Import RATE_GEO_COST table.

```
RATE_GEO_COST
RATE_GEO_COST_SEQ,RATE_GEO_COST_GROUP_GID,CHARGE_AMOUNT,CHARGE_CURRENCY_GID,
CHARGE_AMOUNT_BASE,CHARGE_UNIT_UOM_CODE,CHARGE_UNIT_COUNT,CHARGE_MULTIPLIER,
CHARGE_MULTIPLIER_SCALAR,CHARGE_ACTION,DOMAIN_NAME
1,MYDOMAIN.194-064-3,0.07,USD,0.07,LB,1,SHIPMENT.WEIGHT,,A,MYDOMAIN
2,MYDOMAIN.194-064-3,-50,USD,-50,,1,,,A,MYDOMAIN
3,MYDOMAIN.194-064-3,,,,,1,,65,D,MYDOMAIN
```

Importing TL Rates

This section describes how to specify TL rates and gives sample CSV files for several scenarios.

The following tables must be loaded (in order):

- RATE_OFFERING (setup manually on Oracle Transportation Management web pages)
- X_LANE (see the **Importing X_LANE Data for Rates** section.)

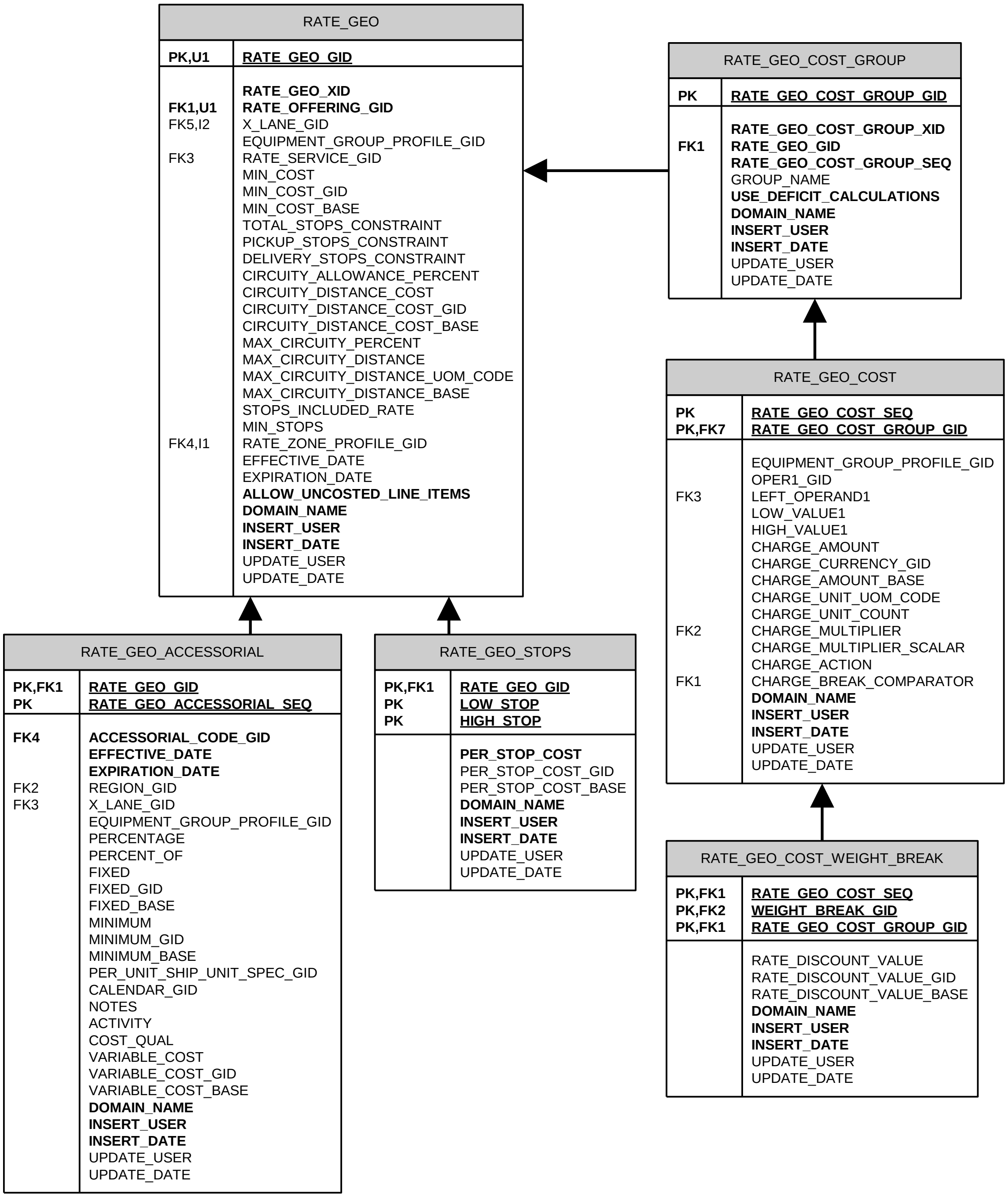
- RATE_GEO
- ACCESSORIAL_CODE
- ACCESSORIAL_COST
- RATE_GEO_ACCESSORIAL (*)
- RATE_GEO_STOPS (*)
- RATE_GEO_COST_GROUP
- RATE_GEO_COST

Note: (*) RATE_GEO_ ACCESSORIAL and RATE_GEO_STOPS must come after RATE_GEO, but are not required before the remaining tables.

Assumptions:

- You have loaded the rate offering table using Oracle Transportation Management web pages
- You have loaded the X_Lane table (see the **Importing X_LANE Data for Rates** section).

Simplified ERD for TL Rates



	RATE_GEO
PK,U1	**RATE_GEO_GID**
	RATE_GEO_XID
FK1,U1	**RATE_OFFERING_GID**
FK5,I2	X_LANE_GID
	EQUIPMENT_GROUP_PROFILE_GID
FK3	RATE_SERVICE_GID
	MIN_COST
	MIN_COST_GID
	MIN_COST_BASE
	TOTAL_STOPS_CONSTRAINT
	PICKUP_STOPS_CONSTRAINT
	DELIVERY_STOPS_CONSTRAINT
	CIRCUITY_ALLOWANCE_PERCENT
	CIRCUITY_DISTANCE_COST
	CIRCUITY_DISTANCE_COST_GID
	CIRCUITY_DISTANCE_COST_BASE
	MAX_CIRCUITY_PERCENT
	MAX_CIRCUITY_DISTANCE
	MAX_CIRCUITY_DISTANCE_UOM_CODE
	MAX_CIRCUITY_DISTANCE_BASE
	STOPS_INCLUDED_RATE
	MIN_STOPS
FK4,I1	RATE_ZONE_PROFILE_GID
	EFFECTIVE_DATE
	EXPIRATION_DATE
	ALLOW_UNCOSTED_LINE_ITEMS
	DOMAIN_NAME
	INSERT_USER
	INSERT_DATE
	UPDATE_USER
	UPDATE_DATE

	RATE_GEO_COST_GROUP
PK	**RATE_GEO_COST_GROUP_GID**
	RATE_GEO_COST_GROUP_XID
FK1	**RATE_GEO_GID**
	RATE_GEO_COST_GROUP_SEQ
	GROUP_NAME
	USE_DEFICIT_CALCULATIONS
	DOMAIN_NAME
	INSERT_USER
	INSERT_DATE
	UPDATE_USER
	UPDATE_DATE

	RATE_GEO_COST
PK	**RATE_GEO_COST_SEQ**
PK,FK7	**RATE_GEO_COST_GROUP_GID**
	EQUIPMENT_GROUP_PROFILE_GID
	OPER1_GID
FK3	LEFT_OPERAND1
	LOW_VALUE1
	HIGH_VALUE1
	CHARGE_AMOUNT
	CHARGE_CURRENCY_GID
	CHARGE_AMOUNT_BASE
	CHARGE_UNIT_UOM_CODE
	CHARGE_UNIT_COUNT
FK2	CHARGE_MULTIPLIER
	CHARGE_MULTIPLIER_SCALAR
	CHARGE_ACTION
FK1	CHARGE_BREAK_COMPARATOR
	DOMAIN_NAME
	INSERT_USER
	INSERT_DATE
	UPDATE_USER
	UPDATE_DATE

	RATE_GEO_ACCESSORIAL
PK,FK1	**RATE_GEO_GID**
PK	**RATE_GEO_ACCESSORIAL_SEQ**
FK4	**ACCESSORIAL_CODE_GID**
	EFFECTIVE_DATE
	EXPIRATION_DATE
FK2	REGION_GID
FK3	X_LANE_GID
	EQUIPMENT_GROUP_PROFILE_GID
	PERCENTAGE
	PERCENT_OF
	FIXED
	FIXED_GID
	FIXED_BASE
	MINIMUM
	MINIMUM_GID
	MINIMUM_BASE
	PER_UNIT_SHIP_UNIT_SPEC_GID
	CALENDAR_GID
	NOTES
	ACTIVITY
	COST_QUAL
	VARIABLE_COST
	VARIABLE_COST_GID
	VARIABLE_COST_BASE
	DOMAIN_NAME
	INSERT_USER
	INSERT_DATE
	UPDATE_USER
	UPDATE_DATE

	RATE_GEO_STOPS
PK,FK1	**RATE_GEO_GID**
PK	**LOW_STOP**
PK	**HIGH_STOP**
	PER_STOP_COST
	PER_STOP_COST_GID
	PER_STOP_COST_BASE
	DOMAIN_NAME
	INSERT_USER
	INSERT_DATE
	UPDATE_USER
	UPDATE_DATE

	RATE_GEO_COST_WEIGHT_BREAK
PK,FK1	**RATE_GEO_COST_SEQ**
PK,FK2	**WEIGHT_BREAK_GID**
PK,FK1	**RATE_GEO_COST_GROUP_GID**
	RATE_DISCOUNT_VALUE
	RATE_DISCOUNT_VALUE_GID
	RATE_DISCOUNT_VALUE_BASE
	DOMAIN_NAME
	INSERT_USER
	INSERT_DATE
	UPDATE_USER
	UPDATE_DATE

Table Notes

RATE_GEO Table

- Allow_uncosted_line_items in Y/N (defaults to "N")

RATE_GEO_ACCESSORIAL

- Left_Operand1 – Basis options define what variable you want to base your conditional charge on.
- Oper1_gid – The operand you compare with.
- Low_value1 – Depending on the operand you use, you might need only the low_value1 or additionally the high_value1.

RATE_GEO_COST_GROUP Table

- Use_deficit_calculations in Y/N (defaults to "N")

RATE_GEO_COST Table

- Oper1_gid – field value "BETWEEN" is a shortcut for X > low and X <= high. Other possible values include "<", "<=", ">", ">=", "=", and "<>".
- charge_unit_uom_code - unit of measure (e.g. "LB" for pounds, or "MI" for miles)
- charge_unit_count - hundredweight, etc.
- charge_action – add (A), setmin (M), setmax (X), multiply (D)
- charge_break_comparator -identifies data element used to access the break

Scenario–Based on Distance Bands with Fixed Charges, and Stop Offs

This scenario assumes that:

- TL rates are defined using distance bands, with a flat charge within each band
- For Rate Geo A

If distance between 10 and 100 miles, charge $50

If distance is between 100 and 200 miles, charge $75

- For Rate Geo B

If distance between 10 and 100 miles, charge $80

1. Import RATE_GEO table.

```
RATE_GEO
RATE_GEO_GID,RATE_GEO_XID,RATE_OFFERING_GID,MIN_COST,MIN_COST_GID,MIN_COST_B
ASE,X_LANE_GID,TOTAL_STOPS_CONSTRAINT,STOPS_INCLUDED_RATE,DOMAIN_NAME
MYDOMAIN.194-064-TL1,194-064-TL1,MYDOMAIN.YELLOW,1,USD,1,MYDOMAIN.194-
064,6,2,MYDOMAIN
MYDOMAIN.194-065-TL1,194-065-TL1,MYDOMAIN.YELLOW,1,USD,1,MYDOMAIN.194-
065,6,2,MYDOMAIN
```

2. Import RATE_GEO_STOPS table.

```
RATE_GEO_STOPS
RATE_GEO_GID,LOW_STOP,HIGH_STOP,PER_STOP_COST,PER_STOP_COST_GID,PER_STOP_COS
T_BASE,DOMAIN_NAME
"MYDOMAIN.194-064-TL1",1,2,50.00,"USD",50.00,"MYDOMAIN"
"MYDOMAIN.194-064-TL1",3,4,100.00,"USD",100.00,"MYDOMAIN"
"MYDOMAIN.194-065-TL1",1,2,25.50,"USD",25.50,"MYDOMAIN"
"MYDOMAIN.194-065-TL1",3,4,85.00,"USD",85.00,"MYDOMAIN"
```

3. Import RATE_GEO_COST_GROUP table.

```
RATE_GEO_COST_GROUP
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_GROUP_XID,RATE_GEO_GID,RATE_GEO_COST_G
ROUP_SEQ,GROUP_NAME,DOMAIN_NAME
"MYDOMAIN.194-064-TL1","194-064-TL1","MYDOMAIN.194-064-
TL1",1,"MY_GROUP_NAME_TL","MYDOMAIN"
"MYDOMAIN.194-065-TL1","194-065-TL1","MYDOMAIN.194-065-
TL1",1,"MY_GROUP_NAME_TL","MYDOMAIN"
```

4. Import RATE_GEO_COST table.

```
RATE_GEO_COST
RATE_GEO_COST_SEQ,RATE_GEO_COST_GROUP_GID,OPER1_GID,LEFT_OPERAND1,LOW_VALUE1
,HIGH_VALUE1,CHARGE_AMOUNT,CHARGE_CURRENCY_GID,CHARGE_AMOUNT_BASE,DOMAIN_NAM
E
1,"MYDOMAIN.194-064-TL1","BETWEEN","SHIPMENT.DISTANCE","10 MI","100
MI",50.00,"USD", 50.00,"MYDOMAIN"
2,"MYDOMAIN.194-064-TL1","BETWEEN","SHIPMENT.DISTANCE","100 MI","200
MI",75.00,"USD", 75.00,"MYDOMAIN"
1,"MYDOMAIN.194-065-TL1","BETWEEN","SHIPMENT.DISTANCE","10 MI","100
MI",80.00,"USD", 80.00,"MYDOMAIN"
```

Scenario–Based on Cost Per Mile, Stop Offs, and Surcharges

This scenario assumes that:

- The freight cost is $1.75 per mile
- Stop Off Charges

Allowed 6 stops total, with 2 stops included in rate

Charge of $50 for 3rd stop, and $65 for subsequent stops

- Fuel Surcharge is $0.02 per mile (Accessorial)
- Minimum charge on transport is $450

Summary

- Total Cost = (distance * 1.75) + stop off charges + (Accessorial of $0.02 per mile)
- Min Transport = (450.00) + stop off charges + (Accessorial of $0.02 per mile)

1. Import RATE_GEO table.

```
RATE_GEO
RATE_GEO_GID,RATE_GEO_XID,RATE_OFFERING_GID,MIN_COST,MIN_COST_GID,MIN_COST_B
ASE,X_LANE_GID,TOTAL_STOPS_CONSTRAINT,STOPS_INCLUDED_RATE,DOMAIN_NAME
"MYDOMAIN.194-064-TL2","194-064-
TL2","MYDOMAIN.YELLOW",1.0,"USD",1.0,"MYDOMAIN.194-064",6, 2,"MYDOMAIN"
```

2. Import ACCESSORIAL_COST table.

```
ACCESSORIAL_COST
ACCESSORIAL_COST_GID,ACCESSORIAL_COST_XID,CHARGE_MULTIPLIER,CHARGE_AMOUNT,CH
ARGE_AMOUNT_GID,CHARGE_AMOUNT_BASE,CHARGE_UNIT_UOM_CODE,CHARGE_UNIT_COUNT,CH
ARGE_ACTION,CHARGE_TYPE,USE_DEFAULTS,CHARGE_MULTIPLIER_OPTION,USES_UNIT_BREA
KS,DOMAIN_NAME,IS_FILED_AS_TARIFF
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
```

```
MYDOMAIN.FS-TL2,FS-
TL2,SHIPMENT.DISTANCE,0.02,USD,0.02,MI,1,A,B,N,A,N,MYDOMAIN,N
```

3. Import ACCESSORIAL_CODE table.

```
ACCESSORIAL_CODE
ACCESSORIAL_CODE_GID,ACCESSORIAL_CODE_XID,ACCESSORIAL_DESC,APPLY_GLOBALLY,DO
MAIN_NAME,IS_FLOW_THRU,IS_VAT_EXEMPT
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FUEL_SURCHARGE,FUEL_SURCHARGE,FUEL SURCHARGE,Y,MYDOMAIN,N,N
```

4. Import RATE_GEO_ACCESSORIAL table.

```
RATE_GEO_ACCESSORIAL
ACCESSORIAL_COST_GID,RATE_GEO_GID,ACCESSORIAL_CODE_GID,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FS-TL2,MYDOMAIN.194-064-TL2,MYDOMAIN.FUEL_SURCHARGE,MYDOMAIN
```

5. Import RATE_GEO_STOPS table.

```
RATE_GEO_STOPS
RATE_GEO_GID,LOW_STOP,HIGH_STOP,PER_STOP_COST,PER_STOP_COST_GID,PER_STOP_COS
T_BASE,DOMAIN_NAME
MYDOMAIN.194-064-TL2,1,1,50,USD,50,MYDOMAIN
MYDOMAIN.194-064-TL2,2,2,65,USD,65,MYDOMAIN
```

Note: Leaving the HIGH_STOP value empty indicates that the last charge will be applied to all the stops greater than the LOW_STOP value. (i.e. for stops >= 2, charge $65 per stop).

6. Import RATE_GEO_COST_GROUP table.

```
RATE_GEO_COST_GROUP
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_GROUP_XID,RATE_GEO_GID,RATE_GEO_COST_G
ROUP_SEQ,GROUP_NAME,DOMAIN_NAME
MYDOMAIN.194-064-TL2,194-064-TL2,MYDOMAIN.194-064-
TL2,1,MY_GROUP_NAME_TL2,MYDOMAIN
```

7. Import RATE_GEO_COST table.

```
RATE_GEO_COST
RATE_GEO_COST_SEQ,RATE_GEO_COST_GROUP_GID,CHARGE_AMOUNT,CHARGE_CURRENCY_GID,
CHARGE_AMOUNT_BASE,CHARGE_UNIT_UOM_CODE,CHARGE_UNIT_COUNT,CHARGE_MULTIPLIER,
CHARGE_MULTIPLIER_SCALAR,CHARGE_ACTION,DOMAIN_NAME
1,MYDOMAIN.194-064-TL2,1.75,USD,1.75,MI,1,SHIPMENT.DISTANCE,,A,MYDOMAIN
2,MYDOMAIN.194-064-TL2,450,USD,450,,1,,,M,MYDOMAIN
```

Note: Seq#2, with a charge action of “M”, indicates that the minimum of the running calculated cost has to be $450 (i.e. if the calculation from Seq#1 is less than $450, then the new value to be used going forward is $450).

An alternative method of specifying this rate would be to recognize that a minimum of $450 equates to distance of 257.143 miles. A comparison for this distance could be used. This would be the corresponding result.

```
RATE_GEO_COST
```

```
RATE_GEO_COST_SEQ,RATE_GEO_COST_GROUP_GID,OPER1_GID,LEFT_OPERAND1,LOW_VALUE1
,CHARGE_AMOUNT,CHARGE_CURRENCY_GID,CHARGE_AMOUNT_BASE,CHARGE_UNIT_UOM_CODE,C
HARGE_UNIT_COUNT,CHARGE_MULTIPLIER,CHARGE_MULTIPLIER_SCALAR,CHARGE_ACTION,DO
MAIN_NAME
1,"MYDOMAIN.194-064-TL2",">","SHIPMENT.DISTANCE","237.143
MI",1.750,"USD",1.750,"MI",1, "SHIPMENT.DISTANCE",,"A","MYDOMAIN"
2,"MYDOMAIN.194-064-TL2","<=","SHIPMENT.DISTANCE","257.143
MI",450.0,"USD",450.0,,1, ,,"A","MYDOMAIN"
```

Note: An alternative to using the data specified for the RATE_GEO_ACCESSORIAL table above would be to add another Sequence to the RATE_GEO_COST table with the following (representing a surcharge of $0.02 per mile):

```
3,"MYDOMAIN.194-064-
TL2",0.020,"USD",0.020,"MI",1,"SHIPMENT.DISTANCE",,"A","MYDOMAIN"
```

Scenario–Based on Cost per Hundredweight, Unit Breaks, and Surcharges

This scenario assumes that:

- The freight cost is per hundredweight based on unit breaks
- Fuel Surcharge is $0.02 per mile (Accessorial)

Summary

- Total Cost = ((weight/100) * (weight break charge)) + (Accessorial of $0.02 per mile)

1. Import RATE_GEO table.

```
RATE_GEO
RATE_GEO_GID,RATE_GEO_XID,RATE_OFFERING_GID,MIN_COST,MIN_COST_GID,MIN_COST_B
ASE,X_LANE_GID,TOTAL_STOPS_CONSTRAINT,STOPS_INCLUDED_RATE,DOMAIN_NAME
MYDOMAIN.194-064-TL3,194-064-TL3,MYDOMAIN.YELLOW,1,USD,1,MYDOMAIN.194-
064,6,2,MYDOMAIN
```

2. Import ACCESSORIAL_COST table.

```
ACCESSORIAL_COST
ACCESSORIAL_COST_GID,ACCESSORIAL_COST_XID,CHARGE_MULTIPLIER,CHARGE_AMOUNT,CH
ARGE_AMOUNT_GID,CHARGE_AMOUNT_BASE,CHARGE_UNIT_UOM_CODE,CHARGE_UNIT_COUNT,CH
ARGE_ACTION,CHARGE_TYPE,USE_DEFAULTS,CHARGE_MULTIPLIER_OPTION,USES_UNIT_BREA
KS,DOMAIN_NAME,IS_FILED_AS_TARIFF
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FS-TL3,FS-
TL3,SHIPMENT.DISTANCE,0.02,USD,0.02,MI,1,A,B,N,A,N,MYDOMAIN,N
```

3. Import ACCESSORIAL_CODE table.

```
ACCESSORIAL_CODE
ACCESSORIAL_CODE_GID,ACCESSORIAL_CODE_XID,ACCESSORIAL_DESC,APPLY_GLOBALLY,DO
MAIN_NAME,IS_FLOW_THRU,IS_VAT_EXEMPT
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FUEL_SURCHARGE,FUEL_SURCHARGE,FUEL SURCHARGE,Y,MYDOMAIN,N,N
```

4. Import RATE_GEO_ACCESSORIAL table.

```
RATE_GEO_ACCESSORIAL
```

```
ACCESSORIAL_COST_GID,RATE_GEO_GID,ACCESSORIAL_CODE_GID,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.FS-TL3,MYDOMAIN.194-064-TL3,MYDOMAIN.FUEL_SURCHARGE,MYDOMAIN
```

5. Import RATE_GEO_COST_GROUP table.

```
RATE_GEO_COST_GROUP
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_GROUP_XID,RATE_GEO_GID,RATE_GEO_COST_G
ROUP_SEQ,GROUP_NAME,DOMAIN_NAME
MYDOMAIN.194-064-TL3,194-064-TL3,MYDOMAIN.194-064-
TL3,1,MY_GROUP_NAME_TL3,MYDOMAIN
```

6. Import RATE_GEO_COST table.

```
RATE_GEO_COST
RATE_GEO_COST_SEQ,RATE_GEO_COST_GROUP_GID,CHARGE_AMOUNT,CHARGE_CURRENCY_GID,
CHARGE_AMOUNT_BASE,CHARGE_UNIT_UOM_CODE,CHARGE_UNIT_COUNT,CHARGE_MULTIPLIER,
CHARGE_MULTIPLIER_SCALAR,CHARGE_ACTION,CHARGE_BREAK_COMPARATOR,DOMAIN_NAME
1,MYDOMAIN.194-064-TL3,,,,LB,100,SHIPMENT.WEIGHT,,A,SHIPMENT.WEIGHT,MYDOMAIN
```

Note: An alternative to using the data specified for the RATE_GEO_ACCESSORIAL table above would be to add another Sequence to this table with the following (representing a surcharge of $0.02 per mile):

```
2,"MYDOMAIN.194-064-
TL3",0.020,"USD",0.020,"MI",1,"SHIPMENT.DISTANCE",,"A","MYDOMAIN"
```

7. Import RATE_UNIT_BREAK_PROFILE table.

```
RATE_UNIT_BREAK_PROFILE
RATE_UNIT_BREAK_PROFILE_GID,RATE_UNIT_BREAK_PROFILE_XID,RATE_UNIT_BREAK_PROF
ILE_NAME,DATA_TYPE,LOOKUP_TYPE,UOM_TYPE,DOMAIN_NAME,INSERT_USER,INSERT_DATE,
UPDATE_USER,UPDATE_DATE
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
"MYDOMAIN.TL 40 TO 45 THOU","TL 40 TO 45 THOU","TL 40 TO 45
THOU","U","M","WEIGHT","MYDOMAIN","MYDOMAIN.ADMIN","20060821190229",,
```

8. Import RATE_UNIT_BREAK table.

```
RATE_UNIT_BREAK
RATE_UNIT_BREAK_GID,RATE_UNIT_BREAK_XID,RATE_UNIT_BREAK_PROFILE_GID,RATE_UNI
T_BREAK_MAX,DOMAIN_NAME,INSERT_USER,INSERT_DATE,UPDATE_USER,UPDATE_DATE
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
"MYDOMAIN.40000","40000","MYDOMAIN.TL 40 TO 45 THOU","40000
LB","MYDOMAIN","MYDOMAIN.ADMIN","20060821190229",,
"MYDOMAIN.45000","45000","MYDOMAIN.TL 40 TO 45 THOU","45000
LB","MYDOMAIN","MYDOMAIN.ADMIN","20060821190229",,
```

9. Import RATE_GEO_COST_UNIT_BREAK table.

```
RATE_GEO_COST_UNIT_BREAK
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_SEQ,RATE_UNIT_BREAK_GID,CHARGE_AMOUNT,
CHARGE_AMOUNT_GID,CHARGE_AMOUNT_BASE,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.194-064-TL3,1,MYDOMAIN.40000,1.14,USD,1.14,MYDOMAIN
MYDOMAIN.194-064-TL3,1,MYDOMAIN.45000,1.07,USD,1.07,MYDOMAIN
```

Scenario–Based on Cost per Hundredweight, Unit Breaks, and Surcharges

This scenario assumes that:

- The freight cost is per hundredweight based on unit breaks which are based on mileage bands.

	Cost per Weight	
Mileage Band	40000 lbs	45000 lbs
0 – 50	0.85	0.50
51 – 55	0.87	0.82
56 - 60	0.88	0.83

- Weighing charge is $20
- Vacuuming fee is $0.25 per CWT with a $115 minimum

Summary

- Total Cost = ((weight/100) * (unit break charge)) + $20 + (Vacuuming Fee of 0.25 per CWT)
- Note: Min $115 for vacuuming is reached when the weight is at 46,000 lbs

1. Import RATE_GEO table.

```
RATE_GEO
RATE_GEO_GID,RATE_GEO_XID,RATE_OFFERING_GID,MIN_COST,MIN_COST_GID,MIN_COST_B
ASE,X_LANE_GID,TOTAL_STOPS_CONSTRAINT,STOPS_INCLUDED_RATE,DOMAIN_NAME
MYDOMAIN.194-064-TL4,194-064-TL4,MYDOMAIN.YELLOW,1,USD,1,MYDOMAIN.194-
064,6,2,MYDOMAIN
```

2. Import RATE_GEO_COST_GROUP table.

```
RATE_GEO_COST_GROUP
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_GROUP_XID,RATE_GEO_GID,RATE_GEO_COST_G
ROUP_SEQ,GROUP_NAME,DOMAIN_NAME
MYDOMAIN.194-064-TL4,194-064-TL4,MYDOMAIN.194-064-
TL4,1,MY_GROUP_NAME_TL4,MYDOMAIN
```

3. Import RATE_GEO_COST table.

```
RATE_GEO_COST
RATE_GEO_COST_SEQ,DOMAIN_NAME,RATE_GEO_COST_GROUP_GID,OPER1_GID,LEFT_OPERAND
1,LOW_VALUE1,CHARGE_AMOUNT,CHARGE_CURRENCY_GID,CHARGE_UNIT_UOM_CODE,CHARGE_U
NIT_COUNT,CHARGE_MULTIPLIER,CHARGE_MULTIPLIER_SCALAR,CHARGE_ACTION,CHARGE_BR
EAK_COMPARATOR,CHARGE_TYPE,CHARGE_MULTIPLIER_OPTION,USES_UNIT_BREAKS,ROUNDIN
G_TYPE,ROUNDING_FIELDS_LEVEL,ROUNDING_APPLICATION,IS_FILED_AS_TARIFF
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
1,MYDOMAIN,MYDOMAIN.194-064-TL4,LT,SHIPMENT.WEIGHT,45000
LB,,,LB,1,SHIPMENT.WEIGHT,,A,SHIPMENT.WEIGHT,B,A,Y,,,,N
2,MYDOMAIN,MYDOMAIN.194-064-TL4,GE,SHIPMENT.WEIGHT,45000
LB,,,LB,1,SHIPMENT.WEIGHT,,A,SHIPMENT.WEIGHT,B,A,Y,,,,N
```

```
3,MYDOMAIN,MYDOMAIN.194-064-TL4,,,,20,USD,,1,SHIPMENT,,A,,B,A,N,N,0,A,Y
```

4. Import RATE_UNIT_BREAK_PROFILE table.

```
RATE_UNIT_BREAK_PROFILE
RATE_UNIT_BREAK_PROFILE_GID,RATE_UNIT_BREAK_PROFILE_XID,DATA_TYPE,LOOKUP_TYP
E,UOM_TYPE,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.LESS THAN 40 PDS,LESS THAN 40 PDS,U,M,WEIGHT,MYDOMAIN
MYDOMAIN.GREATER THAN 45000 PDS,GREATER THAN 45000 PDS,U,M,WEIGHT,MYDOMAIN
```

5. Import RATE_UNIT_BREAK table.

```
RATE_UNIT_BREAK
RATE_UNIT_BREAK_GID,RATE_UNIT_BREAK_XID,RATE_UNIT_BREAK_PROFILE_GID,RATE_UNI
T_BREAK_MAX,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.0-50 MILES,0-50 MILES,MYDOMAIN.GREATER THAN 45000 PDS,45000
LB,MYDOMAIN
MYDOMAIN.51-55 MILES,51-55 MILES,MYDOMAIN.GREATER THAN 45000 PDS,45000
LB,MYDOMAIN
MYDOMAIN.56-60 MILES,56-60 MILES,MYDOMAIN.GREATER THAN 45000 PDS,45000
LB,MYDOMAIN
MYDOMAIN.0-50,0-50,MYDOMAIN.LESS THAN 40 PDS,44999 LB,MYDOMAIN
MYDOMAIN.51-55,51-55,MYDOMAIN.LESS THAN 40 PDS,44999 LB,MYDOMAIN
MYDOMAIN.56-60,56-60,MYDOMAIN.LESS THAN 40 PDS,44999 LB,MYDOMAIN
```

6. Import RATE_GEO_COST_UNIT_BREAK table.

```
RATE_GEO_COST_UNIT_BREAK
RATE_GEO_COST_GROUP_GID,RATE_GEO_COST_SEQ,RATE_UNIT_BREAK_GID,CHARGE_AMOUNT,
CHARGE_AMOUNT_GID,CHARGE_AMOUNT_BASE,CHARGE_DISCOUNT,DOMAIN_NAME
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
MYDOMAIN.194-064-TL4,1,MYDOMAIN.0-50,0.85,USD,0.85,,MYDOMAIN
MYDOMAIN.194-064-TL4,2,MYDOMAIN.51-55,0.87,USD,0.87,,MYDOMAIN
MYDOMAIN.194-064-TL4,3,MYDOMAIN.56-60,0.88,USD,0.88,,MYDOMAIN
MYDOMAIN.194-064-TL4,1,MYDOMAIN.0-50 MILES,0.5,USD,0.5,,MYDOMAIN
MYDOMAIN.194-064-TL4,2,MYDOMAIN.51-55 MILES,0.82,USD,0.82,,MYDOMAIN
MYDOMAIN.194-064-TL4,3,MYDOMAIN.56-60 MILES,0.83,USD,0.83,,MYDOMAIN
```

6. Loading CSV Data via the Application Server

Oracle Transportation Management allows importing of CSV files via the application server. This feature is called "AppServer CSV" or AS.CSV.

If you upload a file whose name ends in "as.csv" instead of just ".CSV", it will be interpreted as an application server CSV file, as opposed to a database-centric CSV file. AppServer CSV files have the following features:

- The first line must be the name of an Entity such as Location, ObOrderBase, OrderRelease, etc. Refer to Example3.java in the chapter titled "Java Integration API" to see how to get a complete list of supported entity names. Entity names are derived from database table names, except they omit the underscores and use mixed case. For example, the entity name for the ob_order_base table is ObOrderBase.
- The second line must be a comma-separated list of attribute names. Attribute names are like database column names, except they omit the underscores and use mixed case. For example, a column called location_gid corresponds to the attribute **locationGid**. Note that the first character is in lower-case for attribute names, but upper case for entity names.
- The third line may be an optional UOM line, which provides UOM values for any UOM attributes. This line may be provided instead of providing UOM qualifiers every time a UOM value occurs.
- The remaining lines are data lines. Each value in a data line must correspond to an attribute name from line2.

  Note: Values for boolean fields should be specified as "true" or "false" rather than "Y" or "N" as with normal csv data data files.

Here is small sample file. This example omits the optional UOM line.

```
Location
locationGid,locationXid,countryCode3Gid,domainName,locationName
"GUEST.MYLOC8","MYLOC8","USA","GUEST","myloc8"
```

Here is another small sample file showing how to specify a UOM line.

```
SShipUnit
domainName,unitWidth,sShipUnitGid,isSplitable,unitNetVolume,unitNetWeight,shipU
nitCount,unitWeight,unitVolume,unitHeight,receivedNetVolume,receivedNetWeight,u
nitLength,sShipUnitXid
UOM:,,,,CUFT,LB,,LB,,,CUFT,LB,,
GUEST,,GUEST.001,false,0,10,1,10,,,0,0,,001
```

Here is the same sample, but with the UOM line omitted and the units of measure specified with each data attribute instead. (Note the use of "false" for the boolean isSplitable field).

```
SShipUnit
domainName,unitWidth,sShipUnitGid,isSplitable,unitNetVolume,unitNetWeight,shipU
nitCount,unitWeight,unitVolume,unitHeight,receivedNetVolume,receivedNetWeight,u
nitLength,sShipUnitXid
GUEST,,GUEST.001,false,0 CUFT,10 LB,1,10 LB,,,0 CUFT,0 LB,,001
```

Here is an example that will result in errors. You cannot specify a UOM line of you also specify UOMs within the data attributes.

> **Note:** The example below represents what not to do. Do NOT copy the example below. The following example would produce an error because a UOM line was specified, but UOMs were also specified in the data attributes. Doing this would cause the system to think that each UOM field has two UOM qualifiers.

```
SShipUnit
domainName,unitWidth,sShipUnitGid,isSplitable,unitNetVolume,unitNetWeight,shipU
nitCount,unitWeight,unitVolume,unitHeight,receivedNetVolume,receivedNetWeight,u
nitLength,sShipUnitXid
UOM:,,,,CUFT,LB,,LB,,,CUFT,LB,,
GUEST,,GUEST.001,false,0 CUFT,10 LB,1,10 LB,,,0 CUFT,0 LB,,001
```

Web Interface for Importing and Exporting AppServer CSV Files

Importing

If you use the Integration Manager to upload a CSV file whose name ends in ".as.csv", Oracle Transportation Management will assume that the content of the file adheres to the rules of AppServer CSV files, and will process it as such. An example of a file name would be "location.as.csv", as opposed to "location.csv".

Each row in the file is processed via the application server instead of directly against the database. This has the benefit of keeping the application server data-cache synchronized with the database.

This page is accessed via **Business Process Automation > Integration > Integration Manager**. See the **Loading CSV Data via Web Pages** chapter for details about this page.

Errors encountered when importing are reported back to the screen.

Exporting

Care must be taken when exporting an AppServer CSV file due to the lack of support for where-clauses. You should be logged in as a user whose vpd_profile limits the number of rows selected from the entity you plan on exporting. Where-clauses will be supported in future releases. In the example below, the user is logged in as "GUEST.FEWROWS". This user has a vpd_profile which limits the number of rows in the s_ship_unit table.

You can use the following URL to export (if it is not on your user menu):

```
http://hostname/servlets/glog.integration.servlet.IntegrationMenuServlet?integr
ation_stylesheet=integration/csvexport.xsl
```

1. In the **command** field, select the "as.xcsv" command.
2. In the "**tableName**" field, specify an "EntityName" instead of a table name. In this case, the entity name is "SShipUnit" which differs from the database table name, which would be "S_SHIP_UNIT".
3. Click the **Run** button. Your output will then appear as follows:

You can then do a "View->Full Screen" in your browser, and select "View Source" (by right-clicking on your mouse). This will place the output in notepad so you can save it to a local file.

Load CSV Files in the Report Owner Directory

Below is the command for loading CSV files in the reportowner directory.

From the application server script8 directory, run the following command.

- `./update_onecsv_rpt.sh REPORT_CONTROL /opt/otm-55-wl/glog/config/dbareportowner`

7. Loading CSV Data via Integration

The GlogXML schema lets you embed the contents of multiple CSV files into a Transmission XML document. The contents of the CSV file are contained in the CSVFileContent XML element within the GLogXMLElement. Only one CSV file can be in a single CSVFileContent XML element. Currently, the interface only supports inserts into the database (corresponds to the 'i' command). The implementation of updates and deletes will be provided in a future release. This interface should only be used for setup activities, and is not intended for operational activity.

GlogXML Document Hierarchy

Below you can see the XML document hierarchy. The elements have been indented to show the hierarchy and relationship.

```
<Transmission>
   <TransmissionHeader> . . .
   </TransmissionHeader>
   <TransmissionBody>
   <GLogXMLElement>
         <CSVFileContent>
               ---CSV File Contents---
         </CSVFileContent>
   </GLogXMLElement>
         <GLogXMLElement>
               <CSVFileContent>
                     ---CSV File Contents---
               </CSVFileContent>
   </GLogXMLElement>
   </TransmissionBody>
</Transmission>
```

Below is a sample document that would be used to insert some data into the rate tables:

```
<Transmission>
<TransmissionHeader>
<UserName>DBA.ADMIN</UserName>
</TransmissionHeader>
<TransmissionBody>
<GLogXMLElement>
<CSVFileContent>
X_LANE
X_LANE_GID,X_LANE_XID,SOURCE_POSTAL_CODE,SOURCE_COUNTRY_CODE3_GID,SOURCE_GEO_HI
ERARCHY_GID,DEST_POSTAL_CODE,DEST_COUNTRY_CODE3_GID,DEST_GEO_HIERARCHY_GID,DOMA
IN_NAME
"MYDOMAIN.194-064","194-
064","194","USA","USZIP3","064","USA","USZIP3","MYDOMAIN"
"MYDOMAIN.194-065","194-
065","194","USA","USZIP3","065","USA","USZIP3","MYDOMAIN"
</CSVFileContent>
</GLogXMLElement>
</TransmissionBody>
</Transmission>
```

-

8. Loading CSV Files as Zip Files

Uploading a Zip File

In addition to the CSV files, your zip file must include a control file called csvutil.ctl to tell Oracle Transportation Management how to process the files. The control file specifies the sequence in which the CSV files should be processed, and specifies the parameters to use when processing each file.

For example, this zip file contains the csvutil.ctl (control) file, and two CSV files, activity.csv and activity2.csv. The csvutil.ctl file contains the following command lines:

```
-dataFileName activity.csv –command 1
-dataFileName activity2.csv –command 1
```

The above control file says to process the file activity.csv using the insert command, then process the file activity2.csv, also using the insert command.

Uploading a zip file is the same as uploading any other file. Use the "Upload an XML/CSV Transmission button accessed via **Business Process Automation > Integration > Integration Manager.**

After uploading your zip file, you are prompted to download a "results" zip file.

Click the **Save** button to save the "results" zip file to your local workstation.

The csvutil.log file in the "result" zip file contains the log from processing all the CSV files in the zip file that you uploaded.

CSV Files that Failed to Load

If any of the records in any of your CSV files fail to load, then your "results" zip file will contain a corresponding ".bad" file containing those records that failed to load. For example, activity.csv.bad and activity2.csv.bad. In this case, both of the CSV files contained one of more records that failed to load, so there is a corresponding .bad file for each CSV file.

Background Zip File Processing

If you are uploading a large zip file, you may want to process your zip file in the background and be notified via email when processing completes. You can then pull your "results" zip file using the "ZipFileDownloadServlet".

For example, this is a "request" zip file whose name specifies that background processing is desired: *test1.bg.zip*. Notice that the filename ends with "bg.zip" rather than just ".zip". This naming convention indicates that background processing is desired. Here is a sample csvutil.ctl file that illustrates how to have an email sent out when processing completes:

```
-dataFileName activity.csv –command 1
-dataFileName activity2.csv –command 1
-mailTo youremail@yourcompany.com –mailFrom youremail@yourcompany.com –subject
zipFileProcessDone –message Hello –smtpHost mail-server.com
```

Clicking on the link in the email takes you to a listing of the zip files on the web server.

You may click on the desired zip file to download it to your local workstation. The zip files ending in "result.zip" are the "results" or "output" zip files.

If things go wrong during background processing, your results zip file will contain a stack trace, which you can read with a text editor rather than WinZip.

9. Exporting CSV Files via the Interface

CSV Export Screens

An initial screen prompts for certain information so the system can determine what additional information is required on subsequent screens.

Exporting Data as a Zip File

This section illustrates how to export a zip file containing one or more CSV files.

1. First, create a csvutil.ctl file containing the commands for exporting your files.

A csvutil.ctl file may contain the following commands:

- `-dataFileName activity_out.csv -command xcsv -tableName ACTIVITY`
- `-dataFileName location_out.csv -command xcsv -tableName LOCATION -whereClause "rownum < 10"`

2. Next, create a zip file containing the csvutil.ct file.
3. Once your zip file is created, you can upload the zip file as you would upload any other file to Oracle Transportation Management:
4. Press the Save button to save the “results” zip file to your local workstation.
5. Open the zip file to see that the zip file contains two CSV files in this case, one corresponding to each command in the csvutil.ct file.
6. The zip file also contains a log file containing information regarding the execution(s) of CSVUtil.

Exporting Large Zip Files in the Background

When exporting a large zip file, you may prefer to export it in the background to avoid the browser timing out. Here is a sample request zip file:

Here are the contents of the csvutil.ctl file within test2.bg.zip:

```
-dataFileName activity_out.csv -command xcsv -tableName ACTIVITY
-dataFileName location_out.csv -command xcsv -tableName LOCATION -whereClause
"rownum < 10"
-mailTo erosenbloom@glog.com -mailFrom erosenbloom@glog.com -subject
zipFileProcessDone -message hello -smtpHost mail-pa.glog.com
```

Here is another example csvutil.ctl file that exports all the rate_geo records in a given domain, along with all parent and child data, but not public data:

```
-dataFileName rate_geo_out.csv -command xcsvwpcd -tableName RATE_GEO -
whereClause "domain_name = 'MDIETL'"
-mailTo erosenbloom@glog.com -mailFrom erosenbloom@glog.com -subject
zipFileProcessDone -message hello -smtpHost mail-pa.glog.com
```

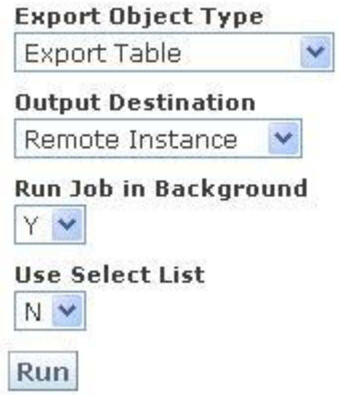
Here is the same example, but this time with referenced public data:

```
-dataFileName rate_geo_out.csv -excludePublic N -command xcsvwpcd -tableName
RATE_GEO -whereClause "domain_name = 'MDIETL'"
-mailTo erosenbloom@glog.com -mailFrom erosenbloom@glog.com -subject
zipFileProcessDone -message hello -smtpHost mail-pa.glog.com
```

Note: Exporting with parent and child data is a very time consuming process since the system has to repeatedly chase after foreign key references. Expect the export to run overnight for as long as 8 hours.

Running CSVUtil in the Background

CSVUtil supports running in the background. The following screen shot shows you how:

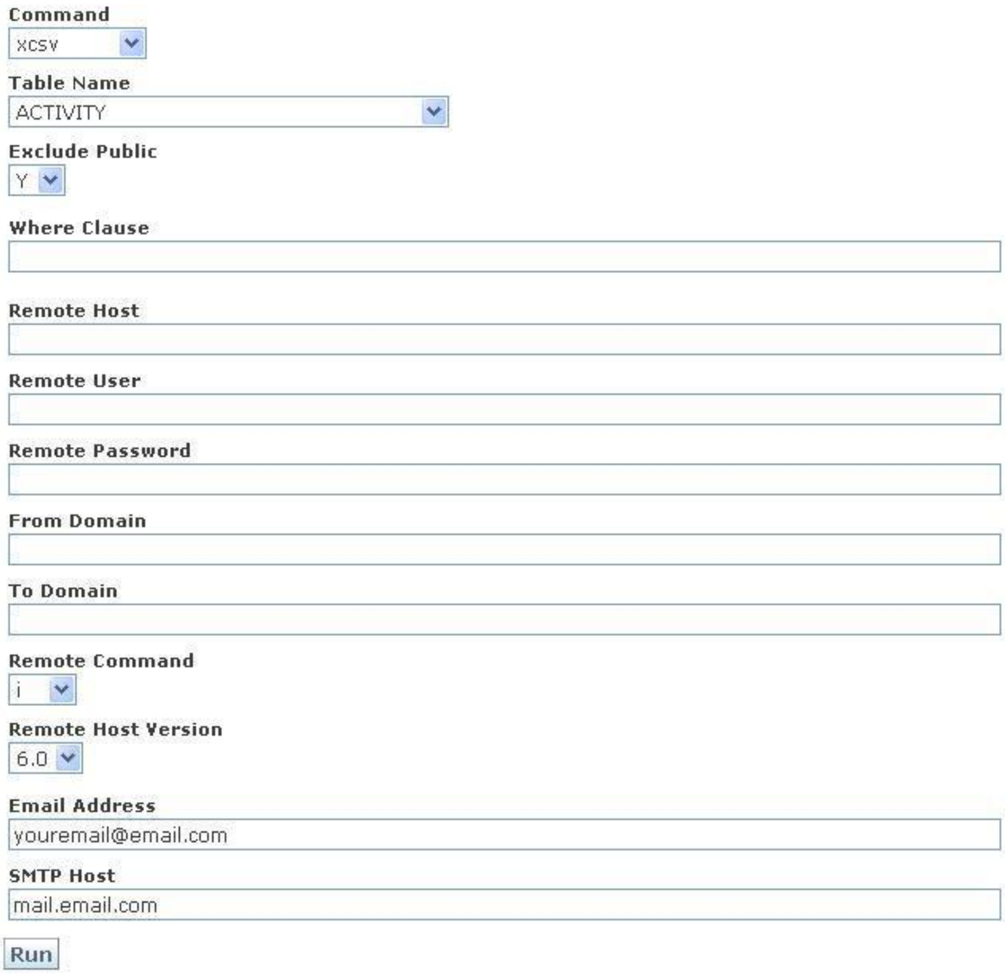


Export Object Type
Export Table

Output Destination
Remote Instance

Run Job in Background
Y

Use Select List
N



Command
xcsv

Table Name
ACTIVITY

Exclude Public
Y

Where Clause

Remote Host

Remote User

Remote Password

From Domain

To Domain

Remote Command
i

Remote Host Version
6.0

Email Address
youremail@email.com

SMTP Host
mail.email.com

As shown above, specify your email address and a SMTP Host to run in the background. The results will be emailed to you when the background job completes (instead of returning the results to the screen).

In this example, the following content was emailed:

```
<CSVUtil>
<Command>xcsv</Command>
<DataDir>/</DataDir>
<DataFileName>null</DataFileName>
<ExcludePublic>true</ExcludePublic>
<Write>
<DatabaseGlobalName>QGC317.HARMONY.GLOGTECH.COM</DatabaseGlobalName>
<Table>ACTIVITY</Table>
<WhereClause>null</WhereClause>
<DomainName>null</DomainName>
<Sql>null</Sql>
<!--
ACTIVITY
ACTIVITY_GID,ACTIVITY_XID,ACTIVITY_NAME,DOMAIN_NAME,INSERT_DATE,UPDATE_DATE,INS
ERT_USER,UPDATE_USER
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYYMMDDHH24MISS'
"RECEIVE","RECEIVE","RECEIVING
FREIGHT","PUBLIC","20011005190259","20021008201735","DBA.ADMIN","DBA.GLOGLOAD"
"LOAD","LOAD","LOADING
FREIGHT","PUBLIC","20011005190259","20021008201735","DBA.ADMIN","DBA.GLOGLOAD"
"LIVELOAD","LIVELOAD","LIVE TRAILER
LOADING","PUBLIC","20011005190259","20021008201735","DBA.ADMIN","DBA.GLOGLOAD"
"DISPATCH","DISPATCH","DRIVER
DISPATCHING","PUBLIC","20011005190259","20021008201735","DBA.ADMIN","DBA.GLOGLO
AD"
"ACTIVATE","ACTIVATE","ITINERARY
ACTIVATED","PUBLIC","20011005190259","20021008201735","DBA.ADMIN","DBA.GLOGLOAD
"
"PICKUP","PICKUP","WAREHOUSE
PICKING","PUBLIC","20011005190259","20021008201735","DBA.ADMIN","DBA.GLOGLOAD"
"CLOSED","CLOSED","WAREHOUSE CLOSED
DOOR","PUBLIC","20011005190259","20021008201735","DBA.ADMIN","DBA.GLOGLOAD"
"OFFICEHOURS","OFFICEHOURS","OFFICE
HOURS","PUBLIC","20011005190259","20021008201735","DBA.ADMIN","DBA.GLOGLOAD"
"BATCH SORT","BATCH SORT","SORTATION AT
DC","PUBLIC","20020125162107","20021008201735","DBA.GLOGLOAD","DBA.GLOGLOAD"
"BATCH DOCK LOAD","BATCH DOCK LOAD","DOCK LOAD AT
DC","PUBLIC","20020125162107","20040308170536","DBA.GLOGLOAD","DBA.ADMIN"
"GUEST.BLAH","BLAH",,"GUEST","20030425012307","20031104125706","DBA.GLOGOWNER",
"DBA.ADMIN"
"RUSHHOURS","RUSHHOURS","RUSH
HOURS","PUBLIC","20030717003037","20040308170536","DBA.ADMIN","DBA.ADMIN"
"GUEST.DLI1","DLI1","DLI1","GUEST","20030717144513",,"GUEST.DLI",
"GUEST.DLI2","DLI2","DLI2","GUEST","20030717144528",,"GUEST.DLI",
"GUEST.TEST","TEST","1","GUEST","20030728200219",,"GUEST.ADMIN",
"GUEST.ABCD","ABCD","VDSFDS","GUEST","20040605190045",,"GUEST.ADMIN",
"GUEST.DTB_SECOND_ACTIVITY","DTB_SECOND_ACTIVITY","DAWN'S SECOND
ACTIVITY","GUEST","20040611120516",,"GUEST.ADMIN",
"GUEST.DTB_FIRST_ACTIVITY","DTB_FIRST_ACTIVITY","DAWN'S FIRST
ACTIVITY","GUEST","20040611120313",,"GUEST.ADMIN",
"GUEST.DTB_NUMBER_3","DTB_NUMBER_3","NUMBER
3","GUEST","20040611121927",,"GUEST.ADMIN",
"ALL","ALL","ALL
ACTIVITIES","PUBLIC","20040910173537","20041213180312","DBA.ADMIN","DBA.ADMIN"
"DEPOT","DEPOT","DEPOT","PUBLIC","20040910173537","20041213180312","DBA.ADMIN",
"DBA.ADMIN"
"OTHER","OTHER","OTHER
ACTIVITIES","PUBLIC","20040921094353","20041213180312","DBA.ADMIN","DBA.ADMIN"
-->
</Write>
</CSVUtil>
```

Normally, you use background processing when initiating lengthy jobs, such as piping a large table set to a RemoteHost.

10. Exporting Referenced PUBLIC Data during Multi-Table Exports

CSVUtil provides the ability to export referenced PUBLIC data during the multi-table export operations (xcsvwcd, xcsvwpd, xcsvwpcd). This feature is especially important when exporting data from a source database where the PUBLIC data has been modified.

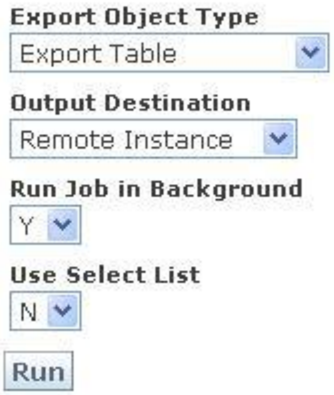
Here is a sample CSVUtil command line for exporting referenced public data:

```
java glog.database.admin.CSVUtil -excludePublic N -command xcsvwpcd -
connectionId localdb4 -dataDir . -dataFileName whatever.csv -tableName RATE_GEO
-whereClause "domain_name = 'DGANO'"
```

Notice the –excludePublic option is set to N, meaning that public data should not be excluded (it should be exported, in other words).

11. Piping CSV Output to a Remote Oracle Transportation Management Instance

CSVUtil supports piping CSV Output to a remote Oracle Transportation Management instance. Refer to the screenshots below:

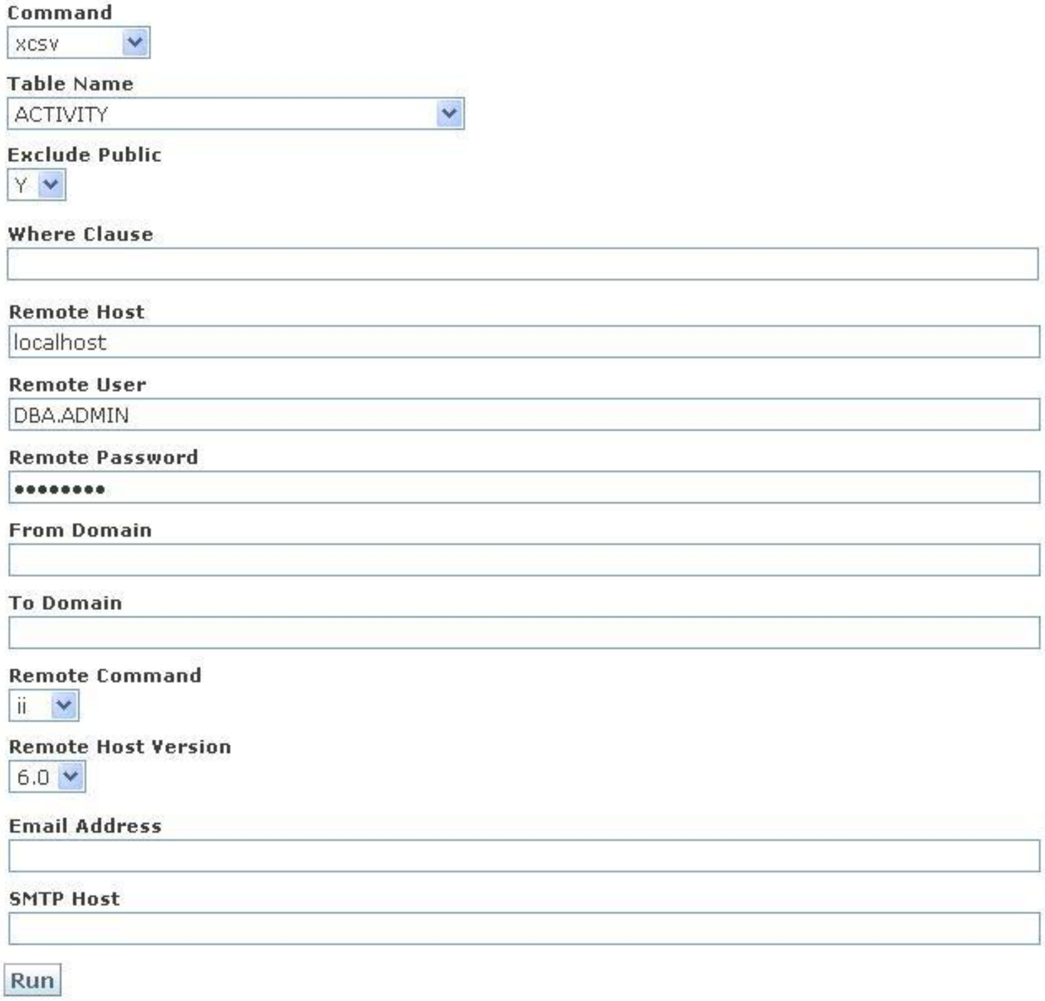


Export Object Type
Export Table

Output Destination
Remote Instance

Run Job in Background
Y

Use Select List
N



Command
xcsv

Table Name
ACTIVITY

Exclude Public
Y

Where Clause

Remote Host
localhost

Remote User
DBA.ADMIN

Remote Password
••••••••

From Domain

To Domain

Remote Command
ii

Remote Host Version
6.0

Email Address

SMTP Host

In the above example, the ACTIVITY table is first exported. The results are then immediately sent to the given Remote Host (in this case back to localhost). You must also specify the Remote User, Remote Password and Remote Command (CSVUtil command) to use on the Remote host.

When you click the Run button you will get XML output showing all the processing that occurred – i.e. export the activity table, send the file over to the remote host, then run CSVUtil on the remote host, and get feedback from the remote host.

Synchronizing Data between Different Oracle Transportation Management Versions

CSVUtil supports the ability to extract and push data to a remote Oracle Transportation Management instance whose version is earlier (or later).

When pushing data to the remote instance, CSVUtil queries the data dictionary to determine which columns in the given table exist on the remote system. Columns which do not exist on the remote system are omitted from the CSV file.

When pushing data to a remote system, you must indicate the version of the remote system. This is required because the format of the URL is different between version 4.x and version 5.x of Oracle Transportation Management.

12. Exporting Table Sets and Piping to a Remote Instance

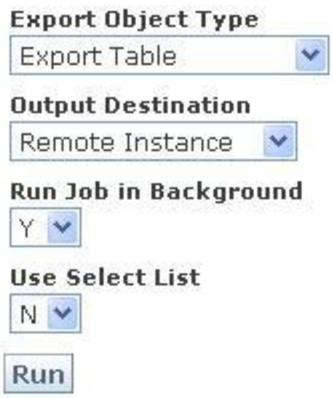
CSVUtil supports exporting ordered table sets. An example of an ordered table set is the EXPORT table set, which lists several hundred tables sorted in foreign key sequence (top-down). Tables in the table_set_detail table may be prefixed by NNNNNNN for the purpose of sequencing the tables. For example:

```
SQL> select table_name from table_set_detail where table_set = 'EXPORT' order
by table_name;

TABLE_NAME
--------------------------------------------------------
0000000.RATE_OPERAND
0001000.ACCESSORIAL_BASIS_PRECEDENCE
0002000.ACCESSORIAL_CODE
0003000.RATABLE_OPERATOR
0004000.RATE_GEO_COST_OPERAND
0005000.COUNTRY_ZONE
0006000.COUNTRY_CODE
0007000.CURRENCY
0008000.DIM_RATE_FACTOR
0009000.ACCESSORIAL_COST
0010000.RATE_UNIT_BREAK_PROFILE
```

As you can see, the tables are prefixed by NNNNNNN in order to ensure they are sequenced within the table set. When you export a table set, you normally pipe it to a remote system. If you do not pipe it to a remote system, it will generate a bunch of temporary files on the source system and leave them there.

Here is a sample screen shot showing how you would normally export the EXPORT table set and pipe it to a remote system.

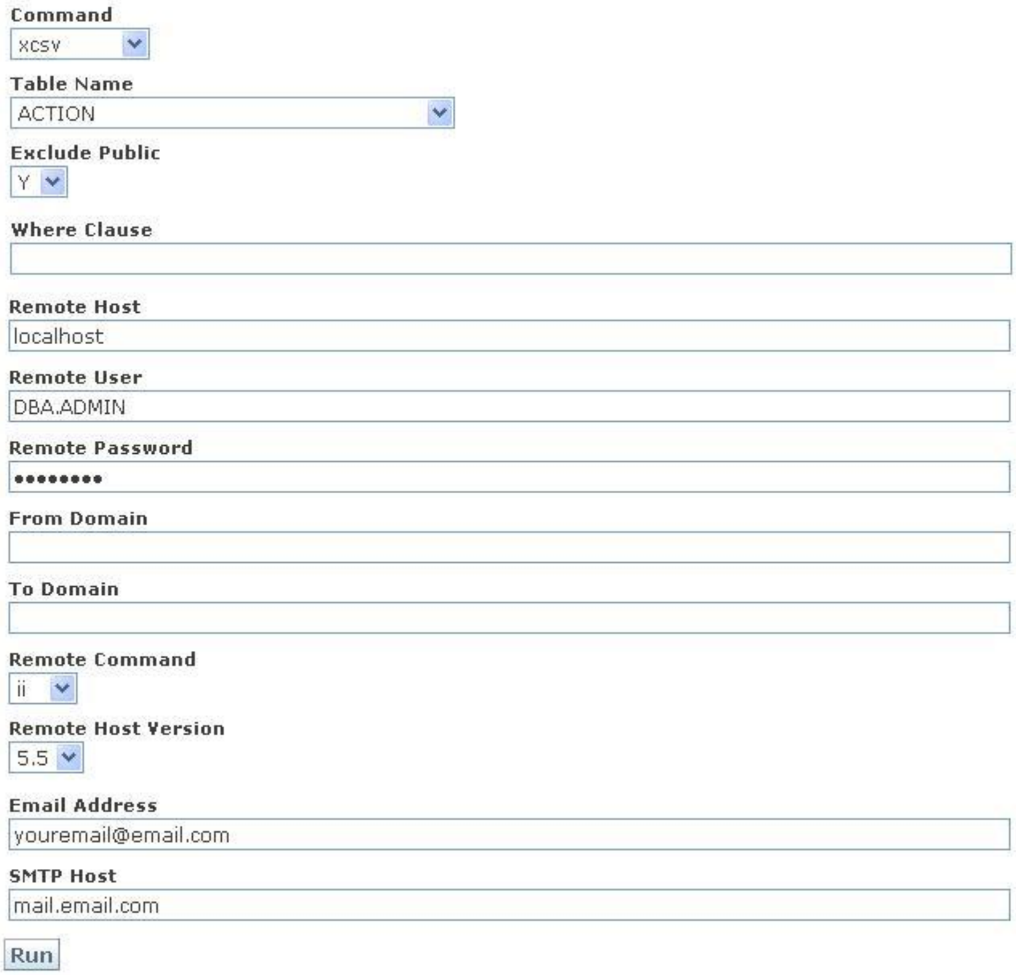


Export Object Type
Export Table

Output Destination
Remote Instance

Run Job in Background
Y

Use Select List
N



Command
xcsv

Table Name
ACTION

Exclude Public
Y

Where Clause

Remote Host
localhost

Remote User
DBA.ADMIN

Remote Password
••••••••

From Domain

To Domain

Remote Command
ii

Remote Host Version
5.5

Email Address
youremail@email.com

SMTP Host
mail.email.com

Notice that the above screen requests background processing by specifying an email address and SMTP Host.

13. Copying Rates between Databases Using Zip Files

CSVUtil can be used to copy a rate_offering, along with all of its prerequisite parent and child data from one database to another.

Step 1 – Create a csvutil.ctl file (CSVUtil Control File) for Exporting

You create a CSVUtil control file containing commands, and then place it in a zip file whose name ends with .bg.zip; for example: exp_rate_offering.bg.zip. When the zip file name ends with "bg.zip", it knows to run the export job in the background. Here are the contents of the csvutil.ctl file to export an entire rate offering:

```
-dataFileName rate_geo_out.csv -command xcsvwpcd -tableName RATE_GEO -
whereClause "rate_offering_gid = 'MDIETL.ASDF'" -excludePublic N
-mailTo erosenbloom@glog.com -mailFrom erosenbloom@glog.com -subject
zipFileProcessDone -message hello -smtpHost mail-pa.glog.com
```

Note: There may only be two lines of text in the above example.

- Place the csvutil.ctl file in a zip file called *name*.bg.zip, where *name* can be anything.
- The xcsvwpcd (export CSV with parent and child data) command will export the rate_geo records, and will recursively export all parent and child records. This can take a while (up to 8 hours).
- The –excludePublic N option means that referenced PUBLIC data will also be exported. If you are sure that your target database has all the required public data, then you can change this to Y, which will save some time on the export.

Step 2 – Use the Integration Upload Screen to Upload the Zip File created in step 1

Use the Integration Upload Screen to upload the exp_rate_offering.bg.zip file. In response to your upload, you immediately receive a message indicating that your export job has been submitted to run in the background. You receive an email when the job completes. The email includes an HTML link to allow you to download the resultant zip file containing your multi-table export.

Step 3 – Download the Zip File Containing the Rate Offering

When you receive the email, download the zip file containing the rate offering, and extract the rate_geo_out.csv file.

Step 4 – Create a csvutil.ctl file for Importing

Similar to step 1, you create another csvutil.ctl file for importing in the background. For example:

```
-dataFileName rate_geo_out.csv -command ii
-mailTo erosenbloom@glog.com -mailFrom erosenbloom@glog.com -subject
zipFileProcessDone -message hello -smtpHost mail-pa.glog.com
```

Step 5 – Create another background zip file

Now create another zip file which will contain the csvutil.ctl file from the previous step, as well as the rate_geo_out.csv file which was exported during step 2. The zip file should again end with "bg.zip".

Step 6 – Upload the zip file from Step 5 to the target instance

To import to the target instance, again use the integration upload screen to upload the background zip file to target instance. You again receive a response indicating that you will get an email when the job completes. The email will again contain a link to allow you to download a results zip file which contains a log file. You will need to examine the log file to see how the import did.

> **Hint**: If you are exporting from a migrated database to a fresh database, use the –removeUndefinedColumns option.

This will tell CSVUtil to ignore deprecated columns.

14. Processing Rate Factors

If you have created rate factors and rate factor rules in Oracle Transportation Management, you can generate accessorial costs in a batch mode fashion. The calculated cost value is placed in the CHARGE_MULTIPLIER_SCALAR column of the Accessorial_Cost table (Charge Discount % field on the Accessorial Cost page).

Process Rate Factors from a Client

You can use a command line utility to process rate factors from a client DOS or UNIX prompt. The command line is managed by a Java class – called the Command Line Processor – available on the OTM servers and so must be run in a shell (DOS or Unix) on one of the web or app servers[2].

The general Command Line Processor format is:

```
java glog.util.command.CommandLineProcessor -command <command name> <…processor
options> <…command specific options>
```

, where the processor options supported by the Rate Factor commands are:

Parameter	**Usage**
`server`	The hostname of the web server where the request will be sent via "http". If non-standard ports are used, the format given should be *hostname:port*.
`username`	OTM application user name to be used for execution of database commands. This is required for the correct VPD security.
`password`	Valid password for "username".

The following table shows the commands available for processing rate factors and the command specific options for each command:

Command Name	**Option(s)**	**Description**
`procRateFactor`	`-rateFactorGid` – The Rate Factor GID.	This will process the specified rate factor using associated rate factor rules. The command selects all rules that refer to that Factor Source GID
`procRateFactorForRule`	`-rateFactorGid` – The Rate Factor GID. `-ruleGid` – The Rate Factor Rule GID.	This will process the specified rate factor using the specified rate factor rule. The command will select the latest rule detail to apply.

---

[2] See Oracle Transportation Management Administration Guide for complete instructions on configuring a Java command environment.

Command Name	Option(s)	Description
`procAllRateFactors`	None.	This will process all unprocessed rate factors using their associated rate factor rules.

Duplicates

The command cannot create duplicates. A duplicate is an accessorial cost with the same Accessorial Code GID and overlapping effective/expiration dates. Take care when setting up the effective and expiration date source logic in the rate factor rule.

Written to Domain

The command generates the accessorial costs in the domain where the rate factor rule exists.

Number of Accessorial Costs

The command generates an accessorial cost for each reference to accessorial default, rate offering, and rate record.

Error Messages

Warning and error messages are logged in the ERROR_LOG table. The following generates errors:

- Inability to calculate the accessorial cost effective/expiration date
- Detected duplicate record
- Unable to calculate accessorial cost value

Undo Changes

There is no specific functionality to undo generated accessorial costs. These tips might help you:

- The way you name your rate factor IDs can help you locate accessorial costs.
- Notes on accessorial costs can help you locate them again. The accessorial cost gets its name according to this template RF_{current date/time}_{first 15 chars of factor source XID}_{effective date of factor value}_{seq num}.

To delete accessorial costs, you need to first delete them from the Accessorial_Default, Rate_Offering_Accessorial, and Rate_Geo_Accessorial record.

15. Importing Voyage Schedule Data

You can import ocean schedules from a variety of portals, like ESG, CargoSmart, INTTRA, and GTNexus.

Assuming you want to load data from multiple providers into separate partitions, load the data from the first provider in the staging tables. Once complete, the data should be moved to the database in the first partition. After the first data set is complete, the data from the second provider should be loaded in the staging tables. After that, the data should be moved to the database in the second partition. This would continue until all the data is loaded.

1. Acquire voyage schedule data. While the data from some providers is available in the correct Oracle Transportation Management format, you need to ensure that the format is correct prior to loading it in the staging tables.
2. Setup Mapping of Data Sources and Partition Keys

   This step is optional, but it makes it easier for you to see what partition key goes with what data provider.

   ```
   pkg_voyage.setup_data_source ('PROVIDER1',1)
   ```

   - PROVIDER1 is the name of the data provider
   - 1 is the partition number.

   Repeat this step to assign data from a second data provider to partition 2 and so on.

   There is a maximum of seven partitions available. It is the VOYAGE and VOYLOC tables that are partitioned, not the staging tables. It is possible to combine multiple data source providers in a single partition. This requires you to load data from all data providers in that partition prior to initiating the loading process.

3. Load the Mapping Tables using normal CSV functionality

Table	**Description**
X_VOY_LOC_MAP	mapping of data source location IDs to Oracle Transportation Management locations GIDs
X_VOY_CAR_MAP	mapping of data source service provider IDs to Oracle Transportation Management Service Provider GIDs

4. Load the Staging Tables using normal CSV functionality

   Load data into the X_VOYAGE and X_VOYLOC tables. The DATA_SOURCE column of the tables should be set to the appropriate data source ID. The data must contain the complete set of voyage data.

5. Delete the current voyage schedules and load the new data set from the staging tables.

   ```
   pkg_voyage.load_schedule (null,200,'Y')
   ```

   - The first parameter is null because the procedure will look up the partition key using the mapping previously setup. If you did not map data sources to partition keys, you need to make sure to load each data provider’s data set in a separate partition.
   - 200 defines the batch size in terms of the number of records the database should hold in its buffer before it writes them to the database
   - ‘Y’ states that errors will be logged to the log file

The Load Schedule procedure takes the new data from the X_VOYAGE and X_VOYLOC tables by cross-referencing service provider IDs with the X_VOY_CAR_MAP table and the location IDs with the X_VOY_LOC_MAP table. Note that if a mapping is missing, the procedure creates a new location using the location ID as the GID, and adds a mapping record to the X_VOY_CAR_MAP or X_VOY_LOC_MAP as needed.

6. View Error Log

```
select * from voyage_err_view
```

If logging was enabled, and there were any problems during the above steps, a message will be posted to the error log.

7. View Data Mappings

If logging was enabled, the current mapping of data source and partition keys can be viewed by executing the following command using a SQL editor:

```
select * from data_source_partition_view
```

Deleting Schedules

pkg_voyage also contains the following:

Purpose	**Procedure**	**Parameters**
Delete all the data in a specified partition.	delete_schedule	p_partkey (PLS_INTEGER)
Delete all the data from a specified data provider. Note that the name of this procedure is the same as the preceding one. The parameter, however, is different.	delete_schedule	p_dataSource (VARCHAR2)

16. Copying Domains

Note: While copying domains, make sure no user accesses the database. You can do this by shutting down the application server.

Note: If you want to copy domain1 that needs data from domain2 and you want domain1 to have access to all data in domain2 in the new target database too, you need to make sure you copy domain2 before domain1.

This chapter describes a set of tools to copy domains. Each of them has its limitations and advantages.

Tool	**Advantages**	**Limitations**	**Usage**
Export/ Import	No physical restrictions on the target and source database servers. For example, they do not need a network link between them.	Tables in your target and source domain must have the same table structure.* Security related tables are not copied. Does not allow you to rename the copied domain name.	Only between databases.
In Schema Copy	Data in CLob and long columns can be copied.	You must rename the copied domain. For tables that have a domain_name column and a numeric primary key, the primary key will increment utilizing sequence numbers. However, its copied child fk column data still points to the old from_domain_name parent.	Only within one database.
Database Link Copy	Preferred tool to build a clean database out of an existing database. Tables in your target and source domain can contain different columns.* You can rename or keep the copied domain name. You can run this script multiple times to insert rows that have been added in the source database since the last database link copy.	Requires that a public database link can be created from the target database to the source database. Allows you to copy CLob and long columns. However, the data types in the local and remote domains must be the same.	Within or between databases.

* Tables might contain different columns if you migrated your source database from an earlier database version and you create your target database with the create_all script. In this case, your migrated database contains obsolete columns since the migration scripts do not generally drop obsolete columns.

Export and Import

This tool exports and imports domain data, child domain data, and referenced domain data. The tool computes the referenced domains from the domain_grant_made table. Furthermore, it copies any table with a domain_name column. You run the domain import/export with two shell scripts.

> **Note:** This tool only works with databases for Oracle Transportation Management version 3.1.1 and later.
>
> **Note:** Only tables that have a domain_name column can be copied.
>
> **Note:** It is crucial to create the target domain before copying data into it.
>
> **Note:** Do not use this tool to copy domains to a production database you plan to go live on. The other tools are better.

What the Objects do

This section describes what each object does.

domain_export.sh

- Calls pkg_domain_export

Searches all the grantor domains that grant read or write access of their domain data to the current domain. However, it does not perform the physical check to see if the current domain does actually reference the grantor domain data. The grantor domain does not include the PUBLIC and SERVPROV domains.

Searches for tables with a domain_name column.

Generates parameter files for export and import

- Calls the Oracle export tool to export the domain data to a dump file.

domain_import.sh

- Calls pkg_domain_export.

Disables some triggers during import.

Disables self-referenced foreign keys during import.

- Calls the Oracle import tool to import the domain data
- Calls pkg_domain_export.

Enables the disabled triggers once the import is completed.

Enables the self-referenced foreign keys once the import is completed.

- Calls pkg_purge and fk_trouble_shooter.

pkg_shipment_purge and fk_trouble_shooter form a backup plan. As mentioned above, PUBLIC and SERVPROV data is not exported. Still, the exported domain might reference PUBLIC or SERVPROV data in the source database leading to foreign key violations. These two packages search for those references and remove them from the target database.

Setup

Compile the packages.

1. Sqlplus> @pkg_shipment_purge.sql
2. Sqlplus> @create_pkg_domain_export.sql
3. Sqlplus> @create_fk_trouble_shooter.sql

Steps to Copy a Domain

Export from source database.

1. Os prompt> `bash domain_export.sh` *`<oracle_sid> <userid> <password> <domainname> <include_reference_domain>`*

   Example: bash domain_export.sh localdb glogowner glogowner guest yes

   The last command line argument, include_reference_domain, is either "yes" or "no". When you enter "yes", the script exports the grantor domain data along with the specified domain data. This is the preferred scenario. However, you can encounter other situations where you export multiple domains and you have already imported the grantor domain data into the target database. If this is the case, enter "no" as the last argument to skip the grantor domains.
2. Review domainexp.log for error messages. You can safely ignore Oracle errors and messages marked EXP-00081.

Import into target database.

3. Create target domain in Oracle Transportation Management.

   If you do not, you will have problems creating the domain in Oracle Transportation Management afterwards. Even if you do create your target domain in Oracle Transportation Management, you will see a lot of error code –1, which means primary key violation. This is okay. The error messages occur because Oracle Transportation Management creates some table data automatically and the copy then tries to insert the same data.

4. Transfer the files domainexp.dmp and domainimp.par to the target database.
5. Os prompt> bash domain_import.sh <*oracle_sid*> <*userid*> <*password*>

Result

When domain_import.sh is done, it displays the message "ALL FOREIGN KEYS WERE ENABLED SUCCESSFULLY!" on the console. If it does not, examine the error logs in domainimp.log and violated_con.log.

- domainimp.log captures all errors during the import.
- violated_con.log gives you more detailed information about constraints. It summarizes all tables with invalid constraints, as well as parent keys, missing in the target database.

Error Messages

The most common problem encountered while importing is foreign key violations, where a large number of rows are rejected. This can be frustrating since it takes a lot of time to display the error messages on the console. Foreign key violations might occur if you migrated your source database from an earlier database version and you created your target database with the create_all script. In this case, your migrated database contains obsolete columns since the migration scripts do not generally drop obsolete columns. To confirm this, search for ORA-00904 error messages in your domainimp.log file.

In Schema Copy

This tool allows you to copy domains within one database. You can copy domains with or without their child domains. Child domains keep their original child domain names; only the parent domain name part is replaced.

What the Objects do

This tool uses these stored procedures in pkg_novpd_domain_copy:

Procedure	**Does This**
set_copy_parameters	Stores what “from_domain” to copy into what “to_domain”. You can enter multiple pairs of domains before executing the actual copying.
print_copy_parameters	Displays the list of “from_domain”s and “to_domain”s you have created with set_copy_parameters.
reset_parameters	Clears the list of domains to copy. You might want to do this if you notice a spelling error.

Set-up

Compile the packages.

1. Log in as glogowner.
2. sqlplus>@create_pkg_novpd_inschema_copy.sql to compile the package.
3. `sqlplus>@novpd_domain_copy_script_builder.sql` to generate the domain copy script, novpd_load.sql. In novpd_load.sql, there is a procedure for every table. Each procedure is enclosed by "declare" and a "/". You can remove a procedure from the script if you do not want to copy a certain table.

Copy Domains

1. sqlplus>execute pkg_novpd_domain_copy.set_copy_parameters('from_domain', 'to_domain', copy_child_domains, domain_info) to set your copy parameters.

   Note: You need to execute this command for every domain to be copied.

   Note: You can copy multiple domains with or without renaming them in a single run.

   Note: If domains depend on each other for data and you want to rename at least one of these domains, you must copy all these domains in a single run. This will allow novpd_load.sql to keep all dependencies correct. If you do not, some of the data will be rejected due to foreign key violations.

Parameter	**Description**
from_domain	Name of the domain to copy.

Parameter	Description
to_domain	The new name of the domain.
copy_child_domains	true - child domains will be copied false - child domains will not be copied.
domain_info	You must enter "false". This retrieves domain information from the local source database instead of a remote target database.

2. sqlplus>set serverout on size 1000000.
3. sqlplus>execute pkg_novpd_domain_copy.print_copy_parameters to display the parameter values entered.
4. If you notice that any of your parameters are wrong, you can reset all parameters with `execute pkg_novpd_domain_copy.reset_parameters`. If you do execute this statement, you must re-enter all your parameters.
5. sqlplus>`@novpd_load.sql` to copy all domains you have entered parameters for. There is a log file: inschema_domain_copy.log.
6. sqlplus>execute domainman.reset_sequence to reset the Oracle sequence numbers.
7. You need to restart Oracle Transportation Management running against the target database to be able to log in to your newly copied domain. The restart allows Oracle Transportation Management to refresh its caches.

Result of In Schema Copy

After novpd_load.sql has finished it displays the number of data rows that were copied and rejected.

Database Link Copy

This is the preferred tool to build a clean database out of an existing database.

Database Link Copy requires the packages pkg_novpd_domain_copy that depends on pkg_domain_export. It copies tables with the domain_name column as well as the security tables.

The total number of rows copied or rejected is written in the database_link_domain_copy.log file. If an exception happens, the exception code as well as the primary key is also written to the log file. Furthermore, if the exception is a foreign key violation, the log will include the foreign key.

Create Link from Target to Source Database

1. Log in to the **target** database with a DBA level account. You must then navigate to the /glog/oracle/script8/ directory.
2. sqlplus>alter system set global_names=false

3. sqlplus>create public database link "loader.oracle.com" connect to "username_in_source_database" identified by " password_in_source_database" using 'source_database'

   Example: create public database link "loader.oracle.com" connect to "glogowner" identified by "glogowner" using 'hera35'

   Use exact double or single quotes as shown above.

   Later, if you need to change the database link to point to a different database, you must first drop the database link (drop public database link loader.oracle.com) and then recreate it.
4. Sqlplus>select count(1) from `shipment@loader.oracle.com` to confirm that the database link is active.

Generate Script

1. Sqlplus>connect glogowner/password@targetdb
2. Sqlplus>@create_pkg_novpd_inschema_copy.sql
3. Sqlplus>`@database_link_domain_copy_script_builder.sql` to generate the link_load.sql script. link_load.sql contains a stored procedure for every table it will copy. Each procedure is enclosed by "declare" and a "/".

   Note: You can remove a procedure from the link_load.sql script if you do not want to copy a certain table. Note that once you remove a procedure for a table, its child tables are rejected.

   Note: Like novpd_load.sql, link_load.sql only contains a stored procedure for tables that the two databases have in common. Furthermore, only data in columns that appear in both the target and source database will be copied. This allows you to copy domains between databases of different releases.

   Note: You may encounter some problems
   1. When uncopied columns are required and have no default values or triggers.
   2. When the same column in both target and source database has different data types such as CLOB and LONG.
   3. When data records in your domain point to records in a domain that do not exist in the target domain. You will see error 2291 in your log file (foreign key violation).
   4. When the sequence number of your source database is higher than your target database. If any of the records in your copied domain refers to a table with only a sequence number as primary key, the referring record will be rejected.

Copy Domains

Note: During the domain copy, only one commit per table and domain is executed. If you want to copy a large amount of data, be sure to allocate enough rollback tablespace and segments.

1. sqlplus>execute pkg_novpd_domain_copy.set_copy_parameters('from_domain', 'to_domain', copy_child_domains, domain_info) to set your copy parameters.

   Note: You need to execute this command for every domain to be copied.

   Note: You can copy multiple domains with or without renaming them in a single run.

   Note: If domains depend on each other for data and you want to rename at least one of these domains, you must copy all these domains in a single run. This will allow link_load.sql to keep all dependencies correct. If you do not, some of the data will be rejected due to foreign key violations.

Parameter	Description
from_domain	Domain name in source database.
to_domain	Domain name in target database. If it is the same as from_domain, then no renaming is performed. Otherwise, from_domain would be renamed to to_domain during the copying.
copy_child_domains	true - then child domains will be copied false - child domains will not be copied.
domain_info	You must enter "true". This retrieves domain information from the remote source database instead of the local target database.

2. sqlplus>set serverout on size 1000000.
3. sqlplus>execute pkg_novpd_domain_copy.print_copy_parameters to display the parameter values entered.
4. If you notice that any of your parameters are wrong, you can reset all parameters with `execute pkg_novpd_domain_copy.reset_parameters`. If you do execute this statement, you must re-enter all your parameters.
5. sqlplus>`@link_load.sql` to copy all domains you have entered parameters for. There is a log file: database_link_domain_copy.log
6. sqlplus>execute domainman.reset_sequence to reset the Oracle sequence numbers.
7. You need to restart Oracle Transportation Management running against the target database to be able to log in to your newly copied domain. The restart allows Oracle Transportation Management to refresh its caches.

Difference Between Domains

You can find the difference between two domains and list the primary keys.

1. Sqlplus>set serverout on size 1000000
2. sqlplus>execute pkg_domain_export.diff_remote(*remote_domain, local_domain*)

   Note: Differences here, most likely depends on static data missing, in your target database, in a domain like PUBLIC. Also, you might have missed to copy dependant domains in one session.

3. sqlplus>execute pkg_domain_export.diff_table_remote(*remote_domain, local_domain*)

Rerun database link copy

As long as the target database schema has not changed, you can run the link_load.sql script again and again to insert rows that have been added to the source database since the last database link copy. This is also useful to keep PUBLIC domains in two databases synchronized. Note that this does not update existing rows in the target database

Note: If the target database schema has changed, you need to run the `database_link_domain_copy_script_builder.sql` script again to create an updated `link_load.sql` script.

17. Deleting Domains

This chapter describes the steps to delete domains in Oracle Transportation Management.

1. Shut down the Oracle Transportation Management application. This includes WebLogic, Tomcat, Apache, etc.
2. Log in directly to the database using a database management utility such as SQLPLUS. Log into the database as `glogowner`.
3. Delete a single domain. Enter the following command at the SQLPLUS prompt:

   ```
   Exec domainman.delete_domain('DOMAIN');
   ```

   Note: Substitute the domain name that you want to delete for DOMAIN. Since this does a cascade delete, this may take a significant amount of time. If there is any data cross-referenced between domains, the data referenced will not be deleted. For example, if Shipments in DomainA reference rates in DomainB, and you delete DomainB, rates in DomainB referenced by shipments in DomainA can not be deleted.

4. Delete mutiple domains. Enter the following commands at the SQLPLUS prompt:

   ```
   Exec domainman.mark_domain_for_delete ('DOMAIN', including_sub_domains);
   ```

   Note: Substitute the domain name that you want to delete for DOMAIN. Including_sub_domains equals "true" or "false". If it is "true", then the child domains are also marked for deleted. Otherwise, the child domains are not included for deletion. This procedure should be called for each domain to be deleted. Every time the procedure is called, the domain and its child domains are cached in memory. If you make a mistake, you have to log out the session and re-log in.

   ```
   Exec domainman.delete_marked_domains;
   ```

   Note: This procedure iterates through all the domains and child domains marked in the previous step. It deletes one table at a time for the domains and their children. It yields better performance. Futhermore, it can delete cross-referenced data within domains in this transaction.

5. Delete non-existent domain data. Enter the following commands at the SQLPLUS prompt:

   ```
   Set serverout on size 1000000
   Exec domainman.report_unreferenced_domains;
   ```

   Note: This procedure reports all the non-existent domains table by table. The non-existent domains are the ones which are not in domain table. They could be from a bug from previous delete domain procedure or the result from loading a CSV. After reviewing the report generated from the previous step, you can call the next procedure to delete the data in all the tables for the non-existent domains.

   ```
   Exec domainman.delete_unreferenced_domains;
   ```

18. Migration Projects

The high level steps involved in a typical use of the Migration Project feature are as follows.

1. Create an **Export** Migration Project on the **Source** system.
2. Define one or more **Migration Objects Group**s in the project.
3. Add one or more **Object ID**s to each Migration Object Group.
4. Export Migration Project. This will create a **Migration Package File.**
5. Request the System Administrator to copy the package file from the Source system to the **Target** system.
6. Run the **Import** process on the Target system to load the data contained in the Migration Package File.

Obviously, for many use cases the above steps are not sufficient; e.g. if the source domain differs from the target domain there must be a step which modifies the content of the package file before import. It is also possible that a configuration is required to be managed by a Version Source Control System and so in theory the package file to be imported could be manually generated and not always the direct result of an Export process.

The following sections describe in detail the steps listed above but also describe sample scenarios which may require additional steps. Any constraints within the current implementation will be described where relevant.

- Creating a Migration Project for Export
- The option also exists to receive the exported package file by email (by subscribing to MIGRATION PROJECT – EXPORT SEND PACKAGE FILE event. See below) or to transport the package via FTP (when Notify Type is specified during the Export Process scheduling).

When sending attachments via Email, the file extension of the package file must be considered. Some email serves do not permit the ".zip" extension for security reasons. By default the Migration Project Package file will use the ".zop" file extension. This can be changed by setting the "glog.migrationproject.export.email.extension" property e.g.

```
glog.migrationproject.export.email.extension=zap
```

However, as the package file could become large it is expected that the use of the file system will be the most effective approach. Sharing of the package with, for example, a Version Control System can be achieved by using a shared file server or mounted directory, dependent on platform.

> **Note**: Automatic check-out/check-in is not configurable directly in OTM.

- Migration Project Package File Contents
- Importing a Migration Project Package

Creating a Migration Project for Export

Migration Project Overview

> **Note**: This section should be read in conjunction with the online help as this will describe the relevant fields available in each UI screen.

By default, new migration projects created via the UI under *Business Process Automation >Migration Project Management > Migration Projects*, will have a **Project Type** of **Export**. The project type is purely informational in this version of the feature; i.e. there is no restriction on exporting projects which have a project type of **Import**. The main use is to identify the primary purpose of the migration project.

The **Manual Sequencing** flag indicates whether or not to allow the application to decide the order in which data is loaded. If this is checked, you must order the data in the required sequence. See the Manual Sequencing section below for a full description.

The **Commit Scope** indicates which transaction scope should be used when importing data. There are two possible values:

- **Object ID** (default) specifies that each object within a group should be treated as an individual unit of work. If one object import fails it will not impact other object transactions.
- **Group** specifies that all the objects within a group should be treated as an individual unit of work. If one object in the group fails then the whole group will fail but no other group will be impacted.

The **Refresh Cache** flag indicates whether in memory cache objects should be refreshed after an import. By default this flag is ticked.

The **Raise Lifetime Events** flag indicates whether lifetime events for objects created, modified or removed on import should raise workflow events. By default the flag is not ticked.

Both these flags are also available on the Object Group. The current setting for the Refresh Cache and Raise Lifetime Events flags at the Project header level are used for each newly created Object Group. For example, if both flags are set, any newly created Object Group will also have both flags set. However, subsequently changing the flags on the Project header level will not affect previously created Object Groups; only ones created after the change.

The next step is to define the groups of objects of specific types, e.g. Agent, by using the **New Migration Object** button.

The type of object in the migration object group is determined by the **Screen Set ID** (also known as a Finder Set). A screen set specifies the Finder used to search for objects of a given type. Once the Screen Set ID is selected e.g. **AGENT**, the grid for saving an **Object ID** will be associated to objects of that type, i.e. by using the Find or List icons on the Object ID field, it will present the Finder Query page for the type, i.e. in this example will present an Agent Finder.

One or more IDs can then be added to the groups and saved. The group can also be edited later to remove previously saved Object IDs or to add new Object IDs.

Any number of additional groups can be added including multiple groups of the same type. By default the list of Migration Object Groups will display in the order they were added. It is only when the Migration Project is **Finished** and saved to the database that the true sequence of how the object groups should be processed is calculated.

The **Transaction Code** specifies how the object data should be processed by the **Target** system and can currently be one of:-

- **Insert**: Create new object and report an error if an object with the same ID already exists
- **Insert Ignore**: Create new object only if object does not already exist. Do not error if it does.
- **Insert Update**: Update an existing object or create one if it does not already exist.
- **Delete**: Remove object. Report an error if it does not exist.

Migration Project Status

During the lifecycle of a migration project, the status of the project can change to reflect its current state.

When an Export project is first created it will be in **ACTIVE** status and when it is successfully exported it will be in **EXPORTED** status. Because an Import project is typically created when a Project Package is imported, the initial status will normally be **IMPORTED** for a successfully imported package. The **PARTIAL** status indicates that some objects were imported successfully whereas some others will have failed.

There is also an **ERROR** status that is intended to show that errors occurred during either the import or export process depending on the Project Type and system where the process occurred.

The **RUNNING** status indicates that an Import or Export process is currently in progress and should therefore ultimately change to one of IMPORTED, PARTIAL, EXPORTED or ERROR when complete based on results.

Migration Object Status

As well as status at the Project level, each migration object group and individual Object ID also has a status. If any one of a group's objects has an ERROR status then the overall group status is also ERROR. The Object ID which is in error will also have a **Failure Reason ID** to explain why the error occurred. See below for a list of possible IDs and an explanation of the reason for failure and options for resolution.

Failure Reason ID	**Reason**	**Solution**
MIGR_PROJ_PK_NOT_EXIST	An Object ID in an Export project does not exist	Recreate Object or remove Object ID from Migration Object Group.
DBXML_IMPORT_INTERNAL_ERROR	DBXML has reported an error trying to load an object group.	Review logs (may need to turn on DBXML log ID) and take appropriate corrective action.

Migration Project Version

The **Version** field is designed for informational purposes to reflect which version of an exported project was used for an import. The Export project will start at version "1" and increment each time an update is made to the project. When an exported package is imported into a target system, the version of the project at the time of export will show in the imported project.

Manual Sequencing

Some object IDs, for example LOCATION, can appear as a foreign key on many tables in the OTM data model. Consequently it can appear, based purely on the data dictionary, that there is a circular relationship between certain tables. Clearly this is not the case as the application context prevents such a situation (many foreign keys are optional fields and only used in certain specific cases).

However, what this means is that for certain object types e.g. Service Provider, depending on the level of configuration complexity on a particular instance, that the automatically determined sequence is not sufficient. However, this is likely to be a rare occurrence[3]. For any such cases, the **Manual Sequencing** override flag can be used to control the sequence of importing groups. Use of this flag

---

[3] The only known case is for Agents triggered by custom events. This requires the AGENT_EVENT records to be imported before the associated AGENT records even though the default sequence has AGENT before AGENT_EVENT.

enables the **Link Sequence** field allowing values to be specified for each Migration Object Group in the project. This will cause the entered values to be used when processing the data and these will not be overridden by the application when the project is saved.

The Link Sequence is an integer field and reordering of groups may require updating multiple groups. For example, if groups exist with sequences 1, 2, 3 & 4 and the requirement is to move the current item 2 to follow the current item 3, this would be achieved as follows:

- Renumber group 4 to be group 5
- Renumber group 2 to be group 4
- Save project.

Creating the Migration Project Package

The migration project defines the specific objects that are considered part of the project but it does not contain any of the content associated with each object. It is essentially just a list of unique keys.

Running the Export process (*Business Process Automation* > *Process Management* > *Migration Project Export*) extracts the content for all (valid) project objects and packages it along with a **Project Control File** (project.xml) into a **Migration Project Package** zip file.

The package file name (i.e. preceding the default file extension of ".zip") will default to the Migration Project XID but can be overridden by using the Migration Project Package field.

In an OTM environment with multiple application servers it becomes necessary to specify which server should execute the Export and Import processing. This is to fix the location for the package files to be one server. If this was not done, then if the package is not copied to ALL servers, it is possible that a scheduled Import process could fail.

Therefore, in a Scalability configuration the following steps should be taken:

- Use Cluster Management to define new cluster for Migration Project Processes
- Assign one machine to this cluster.
- Associate the MIGRATION Application Function to the new cluster.
- Restart Application servers.

Note: The physical location on the server where the package file is stored is controlled by the `glog.migrationProject.dir` property and defaults to `<OTM_HOME>/glog/integration/projects`.

The option also exists to receive the exported package file by email (by subscribing to MIGRATION PROJECT – EXPORT SEND PACKAGE FILE event. See below) or to transport the package via FTP (when Notify Type is specified during the Export Process scheduling).

When sending attachments via Email, the file extension of the package file must be considered. Some email serves do not permit the ".zip" extension for security reasons. By default the Migration Project Package file will use the ".zop" file extension. This can be changed by setting the "glog.migrationproject.export.email.extension" property e.g.

```
glog.migrationproject.export.email.extension=zap
```

However, as the package file could become large it is expected that the use of the file system will be the most effective approach. Sharing of the package with, for example, a Version Control System can be achieved by using a shared file server or mounted directory, dependent on platform.

Note: Automatic check-out/check-in is not configurable directly in OTM.

Migration Project Package File Contents

In the current release, the object content is stored in DB.XML format files where there will be one DB.XML for each migration object group.

The DBXML Export by “Migration Entity” capability is used to extract the content for the project package.

The Migration Entity name used is based on the table name associated to the Screen Set ID for each object group.

The content of a sample project package with one LOCATION object group is listed below:

```
$ jar tvf $OTM_HOME/glog/integration/projects/TEST-001.zip
  3380 Tue Jul 30 10:39:54 PDT 2013 LOCATION_1.db.xml
   785 Tue Jul 30 10:39:54 PDT 2013 project.xml
```

The project.xml file is the Project Control File and contains information on the contents of the package (described in MigrationProject.xsd):-

- ID of source project
- Status of source project on export
- Version of source project on export
- Number of object groups in source project
- Number of object groups successfully exported to this package
- Start/End process times for export.
- For each group:-
    - Transaction code
    - Process sequence
    - Number of objects in the source group
    - Number of objects successfully exported to package
    - Object IDs in group.

Content Modification

There may be occasions where some of the data elements in the source system differ from the required corresponding elements in the target system e.g. domain name, email address, external system URL etc.

If this is the case, the relevant DBXML source file must be unpacked from the package, edited and then repackaged using the same file name as in the original package file.

Importing a Migration Project Package

The import process takes a migration project package and uploads the content contained within it.

By default it will create a new migration project in the importing (target) system to record the groups and object IDs of the imported data and the success or failure of each import.

The package file can be uploaded (referred to as **Local**) at the time of submitting the Migration Project Import Process or already resident on the Application server (referred to as **Server**) in which case it must be located in the directory specified by the `glog.migrationProject.dir` property which defaults to `<OTM_HOME>/glog/integration/projects`.

If no Migration Project ID is specified, the import process will generate a new ID which is a combination of the current date and a unique sequence i.e. YYYYMMDD-nnnn.

If an existing Migration Project ID is specified, the current import will replace all groups and objects in that project with the content from the current project package.

There is an option to control how to react if/when an error occurs in processing the content – **Action On Error** – and can be one of:-

- **CONTINUE:** mark group/object as ERROR status and attempt next group.
- **FAIL**: stop processing immediately and mark project as ERROR.

Import Status

If the import process fails for an object ID in a group it will be placed in ERROR status. If one or more object ID in a group is given an ERROR status, then the group itself is also given an ERROR status.

Similarly, if one or more group in the project is in ERROR status then the project status will be one of PARTIAL or ERROR depending on the number of groups in ERROR status. If not all groups are in ERROR status then the project status will be PARTIAL. If all groups are in ERROR status then the project status will be ERROR.

Migration Project Lifetime Events

As the Migration Project changes state throughout its use various lifetime events will be available to drive notification and agent workflow. All events use the **MIGRATION_PROJECT_EVENT** Notify Function ID. These events include (all prefaced with "**MIGRATION PROJECT –**"):-

- CREATED – Export Migration Project has been created
- MODIFIED: Migration Project updated e.g. Object Group or Object PKs added/removed.
- REMOVED: Migration Project deleted.
- EXPORTED: Migration Project successfully exported.
- EXPORT SEND PACKAGE FILE: Migration Project package has been created and will be included as attachment (EMAIL com method is the only supported option).
- EXPORT FAILED: Migration Project exported failed.
- EXPORT PARTIAL: Migration Project exported but not all requested data was exported.
- IMPORTED: Migration Project successfully imported.
- IMPORT FAILED: Migration Project imported failed.
- IMPORT PARTIAL: Migration Project import completed but not all data was imported successfully.

Command Line Tools

The use of migration project package files will require some level of system administrator access to the application servers. Consequently it may be appropriate to delegate the execution of the export and import to an administrator via formal "change requests", etc.

The command line tools described below provide access to all the same features available via the UI and so are suitable for this purpose.

Export – mpExport.sh (Unix) / mpExport.cmd (MS Windows)

The format of the mpExport command is as follows:-

```
mpExport[.sh|.cmd] –app [app server host:port] –username [OTM user]
[…parameters (below)]
```

Parameter	**Description**
`-projectGID {GID}`	[Mandatory] The {GID} value corresponds to the unique Id of the project to be exported
`-packageFileName {name}`	[Optional] The {name} value is specified when a specific file name is to be used to hold the exported package. If no packageFileName parameter is specified the filename will be constructed using the XID of the exported project GID, replacing any spaces with underscore ('_'). The file extension will be '.zip'.

For example,

```
mpExport.sh -app localhost:7001 -username GUEST.ADMIN -projectGID
GUEST.RELEASE_1 -packageFileName /tmp/release1.zip
```

[will be prompted for specified user's password (in this example for GUEST.ADMIN)]

Import – mpImport.sh (Unix) / mpImport.cmd (MS Windows)

The format of the mpExport command is as follows:-

```
mpImport[.sh|.cmd] -app [app server host:port] -username [OTM user]
[…parameters (below)]
```

Parameter	**Description**
`-packageFileName {name}`	[Mandatory] The {name} value specifies the file name containing the Migration Project Package. is to be used to hold the exported package. If no packageFileName parameter is specified the filename will be constructed using the XID of the exported project GID, replacing any spaces with underscore ('_'). The file extension will be '.zip'.
`-projectGID {GID}`	[Optional] The {GID} value corresponds to the unique ID of the project record that will be created (or updated if it already exists) by importing the specified package file.
`-actionOnError {CONTINUE|FAIL}`	[Optional] Specifies action to take when an error occurs processing the package content. Defaults to CONTINUE.

For example,

```
mpImport.sh -app localhost:7001 -username GUEST.ADMIN -packageFileName
/tmp/release1.zip
```

[will be prompted for specified user's password (in this example for GUEST.ADMIN)]

19. Reference A: DB.XML Transaction Codes

When importing db.xml with any of the methods described in this document, there are three transaction codes currently available:

- I - Insert Mode: Only inserts are performed. If the data already exists in the database, you will get primary key errors.
- IU - Insert/Update Mode: Attempts to insert data. If a primary key violation occurs, it updates the data. No delete statements are generated.
- RC - Replace Children Mode: Deletes all child data corresponding to the top level parent, updates the top level parent, and reinserts the child data. This mode allows for a complete replacement of a data object.
- II: Insert/Ignore Dupes: Attempts to insert a new record but only if it does not exist already.
- D: Delete: Deletes the object and all associated child data. **NOTE**: as this is essentially a 'cascade delete' of an object this should only be used with particular care. For example, deleting a LOCATION may result in deleting SHIPMENTs which are related to that LOCATION.

CSVUtil 5.5 supports a "replace children" (rc) command when processing multi-table CSV files. The rc command will recursively delete all child records and re-insert them from the CSV file. This is useful when you want to completely replace the rows that comprise a complex multi-table business object.

The "C." table sets are used to determine the hierarchical parent/child relationships.

For example:

```
TABLE_SET_DETAIL
TABLE_SET,TABLE_NAME
C.GEO_HIERARCHY,GEO_HIERARCHY_DETAIL
C.GEO_HIERARCHY_DETAIL,HNAME_COMPONENT
```

The C.GEO_HIERARCHY table set indicates that the GEO_HIERARCHY_DETAIL table is a child of geo_hierarchy.

The C.GEO_HIERARCHY_DETAIL table set indicates that the HNAME_COMPONENT table is a child of geo_hierarchy_detail.

Examples:

If you submit the following multi-table CSV file with the "rc" command, all rows in the GEO_HIERARCHY_DETAIL table relating to the GUEST.COUNTRY hierarchy would be deleted (since there are none to replace those records in the CSV file).

```
$HEADER
GEO_HIERARCHY_DETAIL
GEO_HIERARCHY_GID,HNAME_COMPONENT_GID,HLEVEL,DOMAIN_NAME,INSERT_USER,INSERT_DAT
E,UPDATE_USER,UPDATE_DATE
GEO_HIERARCHY
GEO_HIERARCHY_GID,GEO_HIERARCHY_XID,RANK,COUNTRY_CODE3_GID,DOMAIN_NAME,INSERT_U
SER,INSERT_DATE,UPDATE_USER,UPDATE_DATE
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS..'
$BODY
GEO_HIERARCHY
"GUEST.COUNTRY","COUNTRY",10,,"GUEST","DBA.ADMIN",2001-08-30
11:01:56.0,"DBA.ADMIN",2005-10-26 14:44:50.0
```

If you submit the following multi-table CSV file with the “rc” command, there will be two records in the geo_hierarchy_detail table relating to the GUEST.COUNTRY hierarchy, regardless of how many rows were there previously.

```
$HEADER
GEO_HIERARCHY_DETAIL
GEO_HIERARCHY_GID,HNAME_COMPONENT_GID,HLEVEL,DOMAIN_NAME,INSERT_USER,INSERT_DAT
E,UPDATE_USER,UPDATE_DATE
GEO_HIERARCHY
GEO_HIERARCHY_GID,GEO_HIERARCHY_XID,RANK,COUNTRY_CODE3_GID,DOMAIN_NAME,INSERT_U
SER,INSERT_DATE,UPDATE_USER,UPDATE_DATE
EXEC SQL ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS..'
$BODY
GEO_HIERARCHY
"COUNTRY","COUNTRY",10,,"PUBLIC","DBA.ADMIN",2001-08-30
11:01:56.0,"DBA.ADMIN",2005-10-26 14:38:33.0
GEO_HIERARCHY_DETAIL
"COUNTRY","COUNTRY_CODE3",1,"PUBLIC","DBA.ADMIN",2001-08-30 11:01:56.0,,
GEO_HIERARCHY_DETAIL
"COUNTRY","CITY",2,"PUBLIC","DBA.ADMIN",2001-08-30 11:01:56.0,,
```

Sample command line:

```
 java glog.database.admin.CSVUtil -command rc -connectionId localdb -dataDir .
-dataFileName geo_hierarchy.csv
```

In version 5.5 and later, the “rc” command is available after you upload a CSV file via the integration manager.

20. Reference B: Specifying Complex Queries

This section shows the SQL query corresponding to the predefined rate_geo database object.

Example of a Complex Query

Use this example to build your own complex queries when no predefined database object exists for the data you want to export.

```
select rate_geo.*,  \
   cursor (select rate_geo_stops.* from rate_geo_stops where
   rate_geo_stops.rate_geo_gid = rate_geo.rate_geo_gid) as rate_geo_stops,  \
   cursor (select rate_geo_accessorial.* from rate_geo_accessorial where
   rate_geo_accessorial.rate_geo_gid = rate_geo.rate_geo_gid) as
   rate_geo_accessorial,  \
   cursor (select rg_special_service.* from rg_special_service where
   rg_special_service.rate_geo_gid = rate_geo.rate_geo_gid) as
   rg_special_service,  \
   cursor (select rg_special_service_accessorial.* from
   rg_special_service_accessorial where
   rg_special_service_accessorial.rate_geo_gid = rate_geo.rate_geo_gid) as
   rg_special_service_accessorial, \
   cursor (select rate_geo_cost_group.*,   \
         cursor (select rate_geo_cost.* ,   \
         cursor (select rate_geo_cost_weight_break.*  \
         from rate_geo_cost_weight_break  \
         where rate_geo_cost_weight_break.rate_geo_cost_seq =
         rate_geo_cost.rate_geo_cost_seq and
         rate_geo_cost_weight_break.rate_geo_cost_group_gid =
         rate_geo_cost.rate_geo_cost_group_gid) as rate_geo_cost_weight_break
         \
   from rate_geo_cost   \
   where rate_geo_cost.rate_geo_cost_group_gid =
   rate_geo_cost_group.rate_geo_cost_group_gid ) as rate_geo_cost   \
   from rate_geo_cost_group   \
   where rate_geo.rate_geo_gid = rate_geo_cost_group.rate_geo_gid) as
   rate_geo_cost_group \
from rate_geo "
```

The main thing to notice is the use of nested cursors to specify a hierarchical query.

21. Reference C: CSVUtil Response Messages

At the completion of processing the command, CSVUtil responds in the form of an XML message. The XML message may contain the following elements:

- Information passed in as input parameters such as the Command, DataDir, and DataFileName
- Information about the contents of the input file such as the TableName and ColumnList
- An Error element identifying the error that was detected.
- Statistics on the success of the message as follows:
  - ProcessCount: The number of rows that were successfully processed
  - ErrorCount: The number of rows where an error was detected
  - Skipcount: The number of rows that were skipped because of duplicate or missing keys. This is only valid when using the ii command which suppresses unique key constraint violations when inserting data, or the uu and dd commands which suppress "no data found" constraint violations when updating/deleting data.

Response Messages with No Errors

Here is an example of a response indicating no errors. In this case, three data rows (based on the ProcessCount element) of the weight_break.csv file were successfully inserted.

```
<CSVUtil>
<Command>i</Command>
<DataDir>.\</DataDir>
<DataFileName>weight_break.csv</DataFileName>
<ProcessCSV>
<TableName>WEIGHT_BREAK</TableName>
<ColumnList>WEIGHT_BREAK_GID,WEIGHT_BREAK_XID,WEIGHT_BREAK_PROFILE_GID,WEIGHT_B
REAK_MAX,WEIGHT_BREAK_MAX_UOM_CODE,WEIGHT_BREAK_MAX_BASE,DOMAIN_NAME</ColumnLis
t>
<ProcessCount>3</ProcessCount>
<ErrorCount>0</ErrorCount>
<SkipCount>0</SkipCount>
</ProcessCSV>
</CSVUtil>
```

The following is an example of the response message typically received when exporting data using the xcsv command.

```
<CSVUtil>
<Command>xcsv</Command>
<DataDir>.\</DataDir>
<DataFileName>weight_break.csv</DataFileName>
<Write>
<TableName>WEIGHT_BREAK</TableName>
</Write>
</CSVUtil>
```

Error Messages

After processing a command, CSVUtil displays a response in the form of an XML message (see the **Loading CSV Data via the Application Server** section). When an error is detected in the processing, the XML message will contain an Error element with the details. The Error XML element indicates the table name, indicates the type of error detected, and lists the data (or row in file) that was being processed when the error occurred.

Below is the error message that Oracle Transportation Management displayed in the procedure (see the **Loading CSV Data via Integration** section). The TableName element indicates the table being processed, the Exception element provides the error message, and the Data element indicates the row being processed. In this case, it indicates that the JUNK table does not exist in the database.

```
<Error>
<TableName>JUNK</TableName>
<Exception>ORA-00942: table or view does not exist
</Exception>
<Data>"Data1","Data2","Data3"</Data>
</Error>
```

Import

This topic describes some common error messages while importing. For each error, there is an explanation of when the message occurs and the action needed to correct the error.

Heading	**Data**
Message:	<Exception> ORA-00942: table or view does not exist
Occurs When:	Table name improperly specified (misspelled or invalid table) on the first line of the CSV file.
Corrective Action:	Verify that the table exists and that the CSV file contains the correct table name.

Heading	**Data**
Message:	<Exception> ORA-00001: unique constraint (GLOGOWNER.PK_WEIGHT_BREAK) violated
Occurs When:	Inserting data with primary keys that are already in the database.
Corrective Action:	Depending on the action desired, one of the following can be used: • If the data should be skipped or ignored, use the ii command to suppress the message. • If the data is intended to be new, change the keys. • If the data is intended to be an update, use the u or uu command.

Heading	**Data**
Message:	<Exception>ORA-02292: integrity constraint (GLOGOWNER.FK_RGCWB_WEIGHT_BREAK_GID) violated - child record found
Occurs When:	During a delete when child records in other tables depend on the key being removed.

Heading	**Data**
Corrective Action:	Delete child records in associated tables before deleting from this table.

Heading	**Data**
Message:	<Error>There are supposed to be 7 columns of data, but I found 6 columns in this line: ["MYDOMAIN.LT 4500","LT 4500","MYDOMAIN.DEFAULT",4500,"LB","MYDOMAIN"]</Error> <Error> <TableName>WEIGHT_BREAK</TableName> <Exception>ORA-01722: invalid number</Exception> <Data>"MYDOMAIN.LT 4500","LT 4500","MYDOMAIN.DEFAULT",4500,"LB","MYDOMAIN"</Data> </Error>
Occurs When:	Missing a column of data in one of the rows.
Corrective Action:	Verify that the data contains the number of fields as indicated in the ColumnList, and that the field formats (string, numeric, date, etc.) are valid.

Heading	**Data**
Message:	<Error> <TableName>WEIGHT_BREAK</TableName> <Exception>ORA-01722: invalid number</Exception> <Data>"MYDOMAIN.LT 4500","LT 4500","MYDOMAIN.DEFAULT",4500,"LB","gung ho","MYDOMAIN"</Data> </Error>
Occurs When:	Trying to insert a string in a numeric field.
Corrective Action:	Verify that the data contains the number of fields as indicated in the ColumnList, and that the field formats (string, numeric, date, etc.) are valid.

Heading	**Data**

Heading	**Data**
Message:	<Error>There are supposed to be 7 columns of data, but I found 1 columns in this line: []</Error> <Error> <TableName>WEIGHT_BREAK</TableName> <Exception>ORA-01400: cannot insert NULL into ("GLOGOWNER"."WEIGHT_BREAK"."WEIGHT_BREAK_GID") </Exception> <Data></Data> </Error>
Occurs When:	The CSV file contains extra blank lines at the end. Oracle Transportation Management considers each blank line to represent a row of data.
Corrective Action:	Remove any extra blank lines at the end of the file.

Heading	**Data**
Message:	<Error> <TableName>WEIGHT_BREAK</TableName> <Exception>ORA-01401: inserted value too large for column</Exception> <Data>"MYDOMAIN.LT 4500","LT 4500","MYDOMAIN.DEFAULT",4500,"LB",4500,"MYDOMAIN123456789012345678901234567890123456789012345678901234567890"</Data> </Error>
Occurs When:	Field length for one of the columns has been exceeded.
Corrective Action:	Limit the length of the input data field value to the appropriate size.

Heading	**Data**

Heading	**Data**
Message:	<Error> <TableName>WEIGHT_BREAK</TableName> <RowsProcssed>0</RowsProcssed> <Data>"MYDOMAIN.LT 4500","LT 4500","MYDOMAIN.DEFAULT",4500,"LB",4500,"MYDOMAIN"</Data> </Error>
Occurs When:	Attempted to delete data where the data does not exist in the table.
Corrective Action:	Validate that the keys being used to delete the data are correct. Could use the dd command to suppress the error message.

Export

This topic describes some common error messages while exporting. For each error there is an explanation of when the message occurs and the action needed to correct the error.

Heading	**Data**
Message:	<CSVUtil> <Command>xcsv</Command> <DataDir>.\</DataDir> <DataFileName>weight_break.csv</DataFileName> <Write> <TableName>WEIGHT_BREAK2</TableName> </Write> </CSVUtil> Caught exception: CSVUtil.SQLException: /CSVUtil.SQLException: (null)/java.sql.SQLException: ORA-00936: missing expression …
Occurs When:	Attempting to export data from a table that does not exist.

Heading	Data
Corrective Action:	Verify table exists and that the CSV file contains the correct table name.

22. Appendix A: Using Python

Starting in version 6.3 of OTM/GTM, the built in support for Python scripts has been discontinued. However, all previous scripts are still available for download and can be freely used and modified for client use. These scripts are not supported under the OTM Support agreement.

CSV Utilities

Importing on the Client Side

This section describes how to use ClientUtil.py to import data into a remote Oracle Transportation Management database.

> **Note:** ClientUtil does not support the multi-table CSV format.

The following example imports data from d:/temp/rate_geo.csv on your PC into a remote Oracle Transportation Management database. Because xvalidate is set to Y, Oracle Transportation Management does not null missing values in the CSV file and Oracle Transportation Management also validates the content of the CSV file. If you need to null certain fields, set xvalidate to N.

```
python ClientUtil.py
-command csvImport
-hostname localhost
-username GUEST.ADMIN
-password CHANGEME
-localDir d:/temp
-localFileName rate_geo.csv
-xvalidate Y
```

> **Note:** You can skip password and rely on IP authentication instead.

Exporting on the Client Side

This section describes how to use ClientUtil.py to export data from a remote Oracle Transportation Management database.

> **Note:** ClientUtil does not export child and parent data for the specified records(s).

Exporting a Table

The following example exports all the RATE_GEO records in the GUEST domain from the database that is connected to the Oracle Transportation Management instance running on a host called localhost. ClientUtil writes the CSV file to myfile.csv in the d:/temp directory.

```
python ClientUtil.py
-command csvExport
-hostname localhost
-username GUEST.ADMIN
-password CHANGEME
-tableName RATE_GEO
-whereClause "DOMAIN_NAME=’GUEST’"
-localDir d:/temp
-localFileName myfile.csv
```

> **Note:** You can skip password and rely on IP authentication instead.

Exporting Data Based on Any Query

The following example exports a CSV file containing just the shipment_gid column from the shipment table for all records in the GUEST domain. ClientUtil writes the CSV file to d:/temp/myfile.csv on your PC.

```
python ClientUtil.py
-command csvQuery
-hostname localhost
-username GUEST.ADMIN
-password CHANGEME
-sqlQuery "select shipment_gid from shipment where domain_name = ‘GUEST’"
-localDir d:/temp
-localFileName myfile.csv
```

DBXML Utilities

There are two main python scripts that support db.xml files:

- **Sql2xml.py**: generates db.xml output from a select statement
- **Xml2sql.py:** imports a db.xml file into the database

There are a number of ways to use these scripts:

- **via ClientUtil.py** supports client-side batch jobs that export and import db.xml from a remote Oracle Transportation Management instance.
- **Directly on the DOS/UNIX command line** when a local SQL*net connection to the database is available.

ClientUtil support for DB.XML

This chapter describes how to use the client-side python script ClientUtil.py to export and import db.xml files from a remote Oracle Transportation Management database. This section assumes you have python installed on your PC. If not, see the Administration Guide on your Oracle Transportation Management CD for installation and configuration instructions.

The main advantage of ClientUtil.py compared to the web-based interface is that it allows you to write client side batch jobs, which pull db.xml data from a remote Oracle Transportation Management instance. This data can be modified as desired, and then imported back to the remote Oracle Transportation Management instance (also using ClientUtil.py).

Note: ClientUtil.py can also export and import CSV files.

Note: If the Python environment is not enabled the response “Service Unavailable - You need to install Python in order to use this functionality” will be returned.

Exporting DB.XML

Similar to how it works via the web screen, there are two methods for exporting:

- By specifying a dbObjectName and whereClause, or
- By specifying a sqlQuery and a rootName

Using Pre-defined Data Objects

Here is the command line for exporting the first RATE_GEO db-object found in the database:

```
python ClientUtil.py -command xmlExport -hostname localhost -username
GUEST.ADMIN -password CHANGEME -dbObjectName RATE_GEO -whereClause "rownum < 2"
-localDir ./ -localFileName rate_geo1.db.xml
```

This example creates the file “rate_geo1.db.xml” in the current working directory.

You need to modify the following arguments specific to your situation:

- Hostname: hostname of remote web server.
- Username: User name used to login to the remote Oracle Transportation Management instance.
- Password: password corresponding to the username.
- WhereClause: SQL whereClause used to limit size of export.
- LocalDir: directory on your PC where output file is written.
- LocalFileName: name of local output file.

What Pre-defined Data Objects Exist?

Refer to the drop-down list on the xmlexport.xsl page to find out what pre-defined data objects currently exist. At this time, the list contains:

- CORPORATION
- LOCATION
- RATE_GEO
- RATE_OFFERING
- AGENT
- AGENT_ACTION
- AGENT_EVENT
- SAVED_QUERY
- SAVED_CONDITION
- USER_MENU_LAYOUT
- MONITOR_PROFILE
- SHIPMENT
- OB_ORDER_BASE

Using a SqlQuery

Here is a sample command line for exporting all the activity records in the database:

```
python ClientUtil.py -command xmlQuery -hostname localhost -username
GUEST.ADMIN -password CHANGEME -sqlQuery "select * from activity"
 -rootName ACTIVITY -localDir ./ -localFileName activity.db.xml
```

The above command creates the `activity.db.xml` file in the current working directory.

You need to modify the following arguments, specific to your situation:

- Hostname
- Username
- Password
- SqlQuery

- RootName
- LocalDir
- LocalFileName

Importing DB.XML

You can use ClientUtil.py to import a client-side db.xml file into a remote Oracle Transportation Management database instance.

Here is a sample command line:

```
python ClientUtil.py -command xmlImport -hostname localhost -username DBA.ADMIN
-password CHANGEME -transactionCode IU -localDir ./ -localFileName rate.db.xml
```

See the **Reference A: DB.XML Transaction Codes** section for possible transactionCodes.

Oracle Transportation Management ignores element names that do not correspond to a database table. This allows you to comment your DB.XML file without affecting what is imported.

Processing Rate Factors

Note: The functions available in this Python script have been as supported Java commands. See section 14 for details.

Process Rate Factors from a Client

You can use the ClientUtil Python script to process rate factors from a client DOS or UNIX prompt. The following example generates accessorial costs for the specified rate factor source GID.

Command options are:

- `python ClientUtil.py -command procRateFactor -hostname <hostname> -username <un> -password <pw> -rateFactorGid <rfg>` to process the specified rate factor using associated rate factor rules. The command selects all rules that refer to that Factor Source GID
- python ClientUtil.py -command procRateFactorForRule -hostname <hostname> -username <un> -password <pw> -rateFactorGid <rfg> -ruleGid <rG> to process the specified rate factor using the specified rate factor rule. The command will select the latest rule detail to apply.
- python ClientUtil.py -command procAllRateFactors -hostname <hostname> -username <un> -password <pw> to process all unprocessed rate factors using their associated rate factor rules.
- python ClientUtil.py -command procRateFactorRunGroup -hostname <hostname> -username <un> -password <pw> -runGroup <id> to process all rate factors in the specified run group with their associated rate factor rules.
- python ClientUtil.py -command viewRateFactorResults -hostname <hostname> -username <un> -password <pw> to view the results of processing the rate factors.

The same processing logic applies to the commands executed via Python is in the equivalent Java commands covered in section 14.

Modifying Rates Using the RateMgmt.py Script

The RateMgmt.py Python script provides functionality to modify rates. More specifically, it makes it extremely easy to modify a large number of rate records simultaneously.

The script requires installation of the following Python modules:

- Python 2.1 or higher
- PyXML 0.6.6
- 4Suite 0.12

The RateMgmt.py script assumes that you have exported the rate records from the database using the currently available DB XML tool. See section 2 for details on how to do this.

Below is an example of a command line for exporting the rate records that have been marked for expiration:

```
ClientUtil.py -command xmlExport -hostname SERVERONE -username USER.ADMIN -
password CHANGEME -dbObjectName RATE_GEO -whereClause "expire_mark_id =
'TEST_MARK_1'" -localDir X:\FOLDER -localFileName MARKRATES.xml
```

In this example, you are exporting all the rate records from the RATE_GEO table that have an ExpireMarkId equal to TEST_MARK_1. This assumes you have previously set the Expire Mark ID for the appropriate records to TEST_MARK_1 in the user interface. For more details on doing that, please reference the online help for expiring rate offerings and rate records.

Typical things the RateMgmt.py script will be used for include:

- Copy Rate Offerings from AAA to BBB, with a new version for a new, upcoming time period
- Update records as follows:

Add XX% (i.e., add 10%) to a set of Rate Records

Add $XX (i.e., add $50) to a set of Rate Records

Typically you will be adding either a fixed amount or a relative amount, and be able to specify the where clause.

Currently, the RateMgmt.py script supports twelve different commands. You can use the script itself to see the format of each command and to see a brief description of each. To do this use the following command:

```
python RateMgmt.py -command <command>
```

For example, to see the format and get information on the changeRateGeoXid command, you would use:

```
Python RateMgmt.py –command changeRateGeoXid
```

The following sections describe each of the supported RateMgmt.py commands in detail.

changeRateGeoXid

This is used to change a RateGeoXid. It also automatically updates the RateGeoCostGroupGid for the child records.

The format for the command line is:

```
python RateMgmt.py -command changeRateGeoXid -oldGid <oldGid> -newXid <xid> -
inFile <infile> -outFile <outfile>
```

Here is a sample command line for changing the RateGeoXid:

```
python RateMgmt.py -command changeRateGeoXid -oldGid GUEST.1234A -newXid 1234B
-inFile in.xml -outFile out.xml
```

In this example, you are changing the RateGeoXid GUEST.1234A in the input XML file in.xml, to GUEST.1234B in the output XML file out.xml.

In practice, this will often be run before rate records are modified. Since you will most likely need to modify the rates before the old ones actually expire, this will create a rate record with a new ID. That way the rate modifications can be done to the new rate record IDs and the data can be imported back into the database without overriding the current rate records.

changeAllRateGeoXid

This is used to change the suffix of all RateGeoXid(s). It also automatically updates the RateGeoCostGroupGid(s) for the child records.

The format for the command line is:

```
python RateMgmt.py -command changeAllRateGeoXid -numChars <num> -newSuffix
<xidSuffix> -inFile <infile> -outFile <outfile>
```

Here is a sample command line for changing all of the RateGeoXids:

```
python RateMgmt.py -command changeAllRateGeoXid -numChars 5 -newSuffix _2002 -
inFile in.xml -outFile out.xml
```

In this example, you are changing all the rate record IDs in the input XML file in.xml, to include _2002 after what they currently are, and posting the results to the output XML file out.xml. The –numChars argument defines the number of characters in new suffix.

In practice, this will be useful for the same reason as explained under changeRateGeoXid.

changeRateOfferingXid

This is used to change the RateOfferingXid for a rate offering.

The format of the command line is:

```
python RateMgmt.py -command changeRateOfferingXid -oldGid <oldGid> -newXid
<xid> -inFile <infile> -outFile <outfile>
```

Here is a sample command line for changing the RateOfferingXid:

```
python RateMgmt.py -command changeRateOfferingXid -oldGid GUEST.1234A -newXid
1234B -inFile in.xml -outFile out.xml
```

In this example, you are changing the RateOfferingXid GUEST.1234A in the input XML file in.xml to GUEST.1234B in the output XML file out.xml.

In practice, this can be run before rate offerings or records are modified. Since you will most likely need to modify rate offering or records before old ones actually expire, this will create a rate offering with a new ID. That way any modifications can be done to the new rate offering IDs and the data can be imported back into the database without overriding the current data.

changeAllRateOfferingXid

This is used to change the suffix of all RateOfferingXid(s).

The format for the command line is:

```
python RateMgmt.py -command changeAllRateOfferingXid -numChars <num> -newSuffix
<xidSuffix> -inFile <infile> -outFile <outfile>
```

Here is a sample command line for changing all the RateOfferingXids:

```
python RateMgmt.py -command changeAllRateOfferingXid -numChars 5 -newSuffix
_2002 -inFile in.xml -outFile out.xml
```

In this example, you are changing all the rate offering IDs in the input XML file in.xml, to include _2002 after what they currently are, in the output XML file out.xml. The –numChars argument defines the number of characters in the new suffix.

In practice, this will be useful for the same reason as explained under changeRateOfferingXid.

removeExpireMarkId

This is used to remove all the data in the EXPIRE_MARK_ID field of the defined records.

The format for the command line is:

```
python RateMgmt.py -command removeExpireMarkId -inFile <infile> -outFile
<outfile>
```

Here is a sample command line for removing the Expire Mark IDs:

```
python RateMgmt.py -command removeExpireMarkId -inFile in.xml -outFile out.xml
```

In this example, you are removing all the data in the EXPIRE_MARK_ID field for the records in the input XML file in.xml and posting the results in the output XML file out.xml.

In practice, this is helpful for when you modify rate records. A common approach would be to update your rate records, then modify your rates. Since most of the new records have copied information from the original rate records, the new rate records may have expiration mark IDs assigned to them. Since you will not want to have your new, modified rate records marked for expiration, you will use this command to remove their mark IDs.

incRateCostByFactor

This is used to increase your rates by the factor specified. For example, if you need to increase your rates by 10%, you would use this command.

The format for the command line is:

```
python RateMgmt.py -command incRateCostByFactor -factor <increase> [-round
<digits>] [-excBreak Y] [-basis <basis>] -inFile <infile> -outFile <outfile> [-
@table_name.column_name columnValue]
```

Here is a sample command line to increase you rates by 10%:

```
python RateMgmt.py -command incRateCostByFactor -factor 1.10 -inFile in.xml -
outFile out.xml
```

In this example, you are increasing the rates in the input XML file in.xml by 10% and posting the results in the XML output file out.xml. Notice that the -factor argument must be typed as 1.10 for a 10% increase.

This command provides additional arguments to:

- Round the number of digits to a specific value. The value must be an integer greater than or equal to zero. The format of this argument is, -round 2 (which round the rate to the nearest cents in USD).
- Exclude the break (weight or unit) records from being changed. The format of the argument is, -excBreak <xxxxx>
- Specify a filter on the cost basis (e.g. SHIPTMENT, EQUIPMENT, SHIPMENT.DISTANCE (from CHARGE_MULTIPLIER column of RATE_GEO_COST table)). The format of the argument is, -basis <xxxxx>
- Filter for more specific fields. The format of the argument is, -@table_name.column_name columnValue

Here is a sample command line using the -basis argument, as well as a specific field filter:

```
python RateMgmt.py -command incRateCostByFactor -factor 1.10 -basis SHIPMENT -
inFile in.xml -outFile out.xml -@RATE_GEO.X_LANE_GID GUEST.PHL_NYC
```

This would only increase the rates for those rate records where the RateGeo Domain Name is equal to GUEST, and the X_LANE is equal to GUEST.PHL.NYC.

incRateCostByAmount

This is used to increase your rates by the amount specified. For example, if you needed to increase your rates by $50, then you would use this command.

The format for the command line is:

```
python RateMgmt.py -command incRateCostByAmount -amount <amount> -inFile
<infile> -outFile <outfile>
```

Here is a sample command line to increase all your rates by $50:

```
python RateMgmt.py -command incRateCostByAmount -amount 50.00 -inFile in.xml -
outFile out.xml
```

In this example, you are increasing all the rates in the input XML file in.xml by $50 and posting the results to the output XML file out.xml. The currency of the cost is not considered in the command.

This command provides additional arguments to:

- Exclude the break (weight or unit) records from being changed. The format of the argument is, -excBreak <xxxxx>

- Specify a filter on the cost basis (e.g. SHIPTMENT, EQUIPMENT, SHIPMENT.DISTANCE (from CHARGE_MULTIPLIER column of RATE_GEO_COST table)). The format of the argument is, -basis <xxxxx>

The format for each of these is the same as described in incRateCostByFactor.

addNewCostRecord

This is used to add a fixed amount as a new RateGeoCost record. You would use this to create a new rate record with the defined rate cost.

The format for the command line is:

```
python RateMgmt.py -command addNewCostRecord -amount <amount> [-currency
<currencyCode>] -inFile <infile> -outFile <outfile>
```

Here is a sample command line for changing the rate cost:

```
python RateMgmt.py -command addNewCostRecord -amount 5.00 -currency USD -inFile
in.xml -outFile out.xml
```

In this example, you are adding a new cost record based on everything in the input XML file in.xml, giving it a rate cost of $5.00, and posting the results to the output XML file out.xml.

removeUserDateFields

This is used to remove all the INSERT_USER, INSERT_DATE, UPDATE_USER, and UPDATE_DATE fields.

The format for the command line is:

```
python RateMgmt.py -command removeUserDateFields -inFile <infile> -outFile
<outfile>
```

Here is a sample command line:

```
python RateMgmt.py -command removeUserDateFields -inFile in.xml -outFile
out.xml
```

In this example, you are taking the input XML file in.xml, removing all the data in the fields listed above, and posting the results to the output XML file out.xml.

removeField

This is used to remove a specific field.

The format for the command line is:

```
python RateMgmt.py -command removeField -inFile <infile> -outFile <outfile> -
fieldName <fieldName>
```

Here is a sample command line for removing a specific field:

```
python RateMgmt.py -command removeField -inFile in.xml -outFile out.xml -
fieldName EXPIRATION_DATE
```

In this example, you are taking the input XML file in.xml, removing the field EXPIRATION_DATE and all is contents, and posting the results to the output XML file out.xml.

changeEffDate

This is used to change the value in the effective date field. The newDate must be in the format "YYYY-MM-DD HH24:MI:SS" including quotes.

The format for the command line is:

```
python RateMgmt.py -command changeEffDate -inFile <infile> -outFile <outfile> -
newDate <newDate>
```

Here is a sample command line for changing the effective date field:

```
python RateMgmt.py -command changeEffDate -inFile in.xml -outFile out.xml -
newDate "2003-09-01 08:00:00"
```

In this example, the effective date field in the input XML file in.xml will be changed to 2003-09-01 08:00:00. The results will be posted to the output XML file out.xml.

changeFieldValue

This is used to change the value of a specified field. If the new value has spaces, then it must be in quotes.

The format for the command line is:

```
python RateMgmt.py -command changeFieldValue -inFile <infile> -outFile
<outfile> -fieldName <fieldName> -newValue <newValue>
```

Here is a sample command line for changing the value of a specific field:

```
python RateMgmt.py -command changeFieldValue -inFile in.xml -outFile out.xml -
fieldName EXPIRATION_DATE  -newValue "2003-09-01 08:00:00"
```

In this example, the expiration date in the XML input file in.xml will be changed to 2003-09-01 08:00:00. The results will be posted to the output XML file out.xml.

23. Appendix B – Java Integration API (Deprecated)

Note: 'Deprecated' means that the feature will still function and be supported but will be removed in a future release.

Oracle Transportation Management provides a callable Java API to allow external developers to write Java programs that maintain data via the application server. This document describes this API.

This chapter introduces the Java Integration API, taking the perspective of an external developer.

The Java Integration API includes the following methods.

```
package glog.integration.clientapi;
import java.util.Iterator;
public interface ClientAPI
{
   public Iterator getEntityNames () throws ClientAPIException;
   public Iterator describeEntity (String entityName) throws
   ClientAPIException;
   public void insert (ValuesObject rowData) throws ClientAPIException;
   public void insertUpdate (ValuesObject rowData) throws ClientAPIException;
   public void update (ValuesObject rowData) throws ClientAPIException;
   public void delete (ValuesObject rowData) throws ClientAPIException;
   public ValuesObject[] execMany (ValuesObject[] commandList) throws
   ClientAPIException;
   public ValuesObject findByPrimaryKey (ValuesObject primaryKey) throws
   ClientAPIException;
   public ValuesObject[] findAll (String entityName) throws ClientAPIException;
   public void close() throws ClientAPIException;
}
```

The following table briefly describes the purpose of each method. Code examples are then provided to illustrate the use of each method.

Method	**Description**
GetEntityNames	Returns an iteration of all supported entity names, such as Location, ObOrderBase, Shipment, etc. Each entity corresponds to an Oracle Transportation Management table, but the name of the entity uses mixed case instead of underscores. See Example3.java
DescribeEntity	Given an entity name, returns an interaction of ValuesObject each of which describes an attribute of the entity. See Example4.java
Insert	Inserts a new row via the application server. See Example1.java
InsertUpdate	Update a row if it exists, otherwise insert a new row. See Example9.java
Update	Updates a row via the application server. See Example2.java
Delete	Deletes a row via the application server. See Example5.java

Method	Description
ExecMany	Process a sequence of operations in a single transaction. See Example7.java
FindByPrimaryKey	Return a ValuesObject corresponding to a given primary key. See Example6.java
FindAll	Return an array of ValuesObjects corresponding to all rows for the given entity. See Example10.java
Close	Close a connection

Example1.java – Insert

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
public class Example1
{
    static public void main(String[] args) throws Exception
    {
            ClientAPI clientAPI =
            ClientAPIConnection.connect("GUEST.ADMIN","CHANGEME");
            ValuesObject rowData = new ValuesObject("Location");
            rowData.put("locationGid","GUEST.MYNEWLOC4");
            rowData.put("locationXid","MYNEWLOC4");
            rowData.put("countryCode3Gid","USA");
            rowData.put("domainName","GUEST");
            rowData.put("isTemporary","true");
            clientAPI.insert(rowData);
    }
}
```

Example2.java – Update

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;

public class Example2
{
    static public void main(String[] args) throws Exception
    {
            ClientAPI clientAPI =
            ClientAPIConnection.connect("GUEST.ADMIN","CHANGEME");
            ValuesObject rowData = new ValuesObject("Location");
            rowData.put("locationGid","GUEST.MYNEWLOC");
            rowData.put("locationName","Eric Rosenbloom");
            clientAPI.update(rowData);
    }
}
```

Example3.java – GetEntityNames

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
import java.util.Iterator;
public class Example3
{
    static public void main(String[] args) throws Exception
    {
            ClientAPI clientAPI =
            ClientAPIConnection.connect("GUEST.ADMIN","CHANGEME");
            Iterator i = clientAPI.getEntityNames();
            while (i.hasNext()) {
                    System.out.println("EntityName = " + (String) i.next());
            }
    }
}
```

Example4.java – DescribeEntity

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
import java.util.Iterator;
public class Example4
{
    static public void main(String[] args) throws Exception
    {
            ClientAPI clientAPI =
            ClientAPIConnection.connect("GUEST.ADMIN","CHANGEME");
            Iterator i = clientAPI.getEntityNames();
            while (i.hasNext()) {
                    String entityName = (String) i.next();
                    System.out.println(entityName);
                    Iterator attributeList = clientAPI.describeEntity(entityName);
                    while (attributeList.hasNext()) {
                            ValuesObject metaData = (ValuesObject)
                            attributeList.next();
                            System.out.println("      " + (String)
                            metaData.get("AttributeName") + " " + (String)
                            metaData.get("DataType"));
                    }
            }
    }
}
```

Example5.java – Delete

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
public class Example5
{
   static public void main(String[] args) throws Exception
   {
          ClientAPI clientAPI =
          ClientAPIConnection.connect("GUEST.ADMIN","CHANGEME");
          ValuesObject primaryKey = new ValuesObject("Location");
          primaryKey.put("locationGid", "GUEST.MYNEWLOC");
          clientAPI.delete(primaryKey);
   }
}
```

Example6.java – FindByPrimaryKey

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
public class Example6
{
   static public void main(String[] args) throws Exception
   {
          ClientAPI clientAPI =
          ClientAPIConnection.connect("MDIETL.ADMIN","CHANGEME");
          ValuesObject primaryKey = new ValuesObject("Shipment");
          primaryKey.put("shipmentGid", "MDIETL.184");
          ValuesObject rowData = clientAPI.findByPrimaryKey(primaryKey);
          System.out.println("rowData = " + rowData);
   }
}
```

Example7.java – ExecMany

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
public class Example7
{
   static public void main(String[] args) throws Exception
   {
          ClientAPI clientAPI =
          ClientAPIConnection.connect("GUEST.ADMIN","CHANGEME");

          // Construct ValuesObject for first update command
          ValuesObject rowData1 = new ValuesObject("Location");
          rowData1.put("locationGid", "GUEST.MYNEWLOC");
          rowData1.put("locationName","My location name");
          ValuesObject update1 = new ValuesObject("update");
          update1.put("rowData", rowData1);

          // Construct ValuesObject for second update command
          ValuesObject rowData2 = new ValuesObject("Location");
          rowData2.put("locationGid", "GUEST.MYNEWLOC2");
          rowData2.put("locationName","My location name2");
          ValuesObject update2 = new ValuesObject("update");
          update2.put("rowData", rowData2);

          // Now execute both update commands as a single transaction.
          // The method returns the commandList that you passed in, with an
          "status" field
          // added to each element to describe the success or failure of each
          command.
          ValuesObject results[] = clientAPI.execMany(new
          ValuesObject[]{update1, update2});

          // print the status of each command
          for (int i = 0; i < results.length; i++) {
                ValuesObject command = results[i];
                String status = (String) command.get("status");
                if (status != null) {
                      System.out.println("status of command " + i + " = " +
                      status );
                      if (status.equals("error")) {
                            String stackTrace = (String)
                            command.get("stackTrace");
                            System.out.println("stackTrace of failed command
                            = " + stackTrace);
                      }
                }
          }
    }
}
```

Example9.java – InsertUpdate

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
public class Example9
{
    static public void main(String[] args) throws Exception
    {
          ClientAPI clientAPI =
          ClientAPIConnection.connect("GUEST.ADMIN","CHANGEME");
          ValuesObject rowData = new ValuesObject("Location");
          rowData.put("locationGid","GUEST.MYNEWLOC4e");
          rowData.put("locationXid","MYNEWLOC4e");
          rowData.put("countryCode3Gid","USA");
          rowData.put("domainName","GUEST");
          rowData.put("isTemporary","true");
          clientAPI.insertUpdate(rowData);
    }
}
```

Example10.java – FindAll

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
public class Example10
{
   static public void main(String[] args) throws Exception
   {
         ClientAPI clientAPI =
         ClientAPIConnection.connect("MDIETL.ADMIN","CHANGEME");
         ValuesObject[] set = clientAPI.findAll("Shipment");
         for (int i = 0; i < set.length; i++) {
               System.out.println(set[i]);
         }
   }
}
```

Example11.java – Exception Handling

All the ClientAPI methods throw ClientAPIException. This example shows how you may catch a ClientAPIException.

```
package glog_deploy.integration.clientapi;
import glog.integration.clientapi.ValuesObject;
import glog.integration.clientapi.ClientAPIConnection;
import glog.integration.clientapi.ClientAPI;
import glog.integration.clientapi.ClientAPIException;
public class Example11
{
   static public void main(String[] args) throws Exception
   {
         // Catch a bad password
         try {
               ClientAPI clientAPI =
               ClientAPIConnection.connect("GUEST.ADMIN","WRONGPASSWORD");
         }
         catch (ClientAPIException cae) {
               cae.printStackTrace(System.out);
         }
   }
}
```

The ClientAPIConnection Class

The ClientAPIConnection class provides a connect() method which authenticates the client application and returns an instance of a class which implements the ClientAPI.

The ValuesObject Class

The ValuesObject class is a thin wrapper around java.util.HashMap, providing support for a set of attribute/value pairs.

Handling Units of Measure

The output from Example6.java can be used to understand how units of measure are represented within the ValuesObject.

java glog_deploy.integration.clientapi.Example6

```
rowData = {isTemperatureControl=false, domainName=MDIETL,
checkCapacityConstraint=true, itineraryGid=MDIETL.180, totalActualCost=2880.44
USD, startTime=2002-09-17 17:47:28 UTC, totalVolume=1000 CUFT,
isCostFixed=false, plannedCost=2880.44 USD, totalWeight=40000 LB,
totalWeightedCost=2880.44 USD, totalNetVolume=1000 CUFT,
isServiceTimeFixed=false, rateGeoGid=MDIETL.CA-GA.MSCARRIERS,
feasibilityCode=FEASIBLE, rateOfferingGid=MDIETL.MSCARRIERS2000, endTime=2002-
09-22 17:47:28 UTC, totalNetWeight=40000 LB, isFixedTenderContact=false,
isTemplate=false, shipmentAsWork=false, numStops=2, checkCostConstraint=true,
weighCode=A, isRateOfferingFixed=false, shipmentReleased=true,
sourceLocationGid=MDIETL.CONTAINER MFG - PLANT 1 - LOS ANGELES,
isAutoMergeConsolidate=false, shipmentTypeGid=TRANSPORT, isToBeHeld=false,
parentLegGid=MDIETL.1, isPreferredCarrier=false, servprovGid=MDIETL.MSCARRIERS,
transportModeGid=TL, destLocationGid=MDIETL.100 INDUSTRIAL ROAD, perspective=B,
shipmentGid=MDIETL.184, totalShipUnitCount=1, shipmentXid=184, rule7=false,
isServprovFixed=false, shipmentName=erosenbloom, numOrderReleases=1,
checkTimeConstraint=true, isPreload=false, isHazardous=false,
isRateGeoFixed=false}
```

The above output illustrates several UOM attributes. For example, the UOM of "totalActualCost" is "USD", and the UOM of "startTime" is "GMT". When writing code such as Example1.java, you must specify a unit of measure for any attribute where a unit of measure makes sense. (A Remark would be an example of an attribute where a unit of measure would not make sense).

The valid UOM codes can be determined by querying the UOM table.

Environment Issues

The ClientAPIConnection class depends on there being a glog.properties file in the user.home directory. This property file is required to determine which application server to connect to. (Notice that only the username and password is specified when you make the connect call from your java program).

Here are the minimal entries required in the glog.properties file:

```
# application server URL and port
appserver=localhost
appserver.port=7001
```

On an NT machine, the above glog.properties file resides in the default user.home directory:

```
c:/WINNT/Profiles/username/glog.properties
```

You can specify user.home on the java command line, and then ClientAPIConnection will find the glog.properties file in the directory you specify. For example:

```
java -Duser.home=l:/gc3/glog_deploy/app/config
glog_deploy.integration.clientapi.Example1
```

In the above example, you tell the JVM that the user.home directory is l:/gc3/glog_deploy/app/config instead of the default c:/WINNT/Profiles/username.