

Warehouse Builder 11g

Extending OWB: Building New ERP Application Connectors

December 2008

Extending OWB: Building New ERP Application Connectors

INTRODUCTION

Included with every Oracle database, OWB provides ETL, data integration and data warehousing capabilities for the Oracle Database 10gR2, 11gR1 and later. Its features are intended to increase developer productivity across all data integration tasks. Beyond basic integration of relational data, OWB allows developers to work with their data at higher levels of abstraction and focus more on the logical objects represented than on the physical realization of that data in database tables.

For data warehousing, this means modeling dimensions and cubes at a logical level, specifying properties for dimensions and cubes and storage types, and allowing OWB to define and manage the physical database objects and accompanying logic.

For ERP or similar applications built on databases, this means filtering out irrelevant database objects to simplify the developer's view of data sources and targets, so that they can focus the business objects represented by the application's data.

The database schema underlying an ERP application can be difficult to work with in designing an ETL process, for a number of reasons:

- Cryptic naming conventions make it difficult to recognize the business objects represented by an application's underlying tables, views, sequences and columns.
- The schema may include numerous tables, views and columns that are used internally and that do not correspond directly to logical-level business objects as presented to the end user. Designing an ETL process requires

knowing which objects to ignore, which are significant, which are safe to extract value from, and which are safe to modify.

- The database schema for complex ERP applications often has a very large number tables, views and sequences, where naming conventions are of limited help in helping ETL developers separate objects according to business area.

To simplify working with the major ERP applications in the market as sources and targets, Oracle Warehouse Builder has offered application connector options for applications. The benefits of application connectors include:

- The database objects exposed to the OWB user can be limited to those that correspond to the logical-level business objects managed by the application.
- Those objects can be managed and used in OWB under meaningful business names instead of cryptic physical-level names, and grouped into business areas if the application provides such metadata.

The result is improved developer productivity on projects where the supported ERP applications are used as sources and/or targets for OWB in data integration, data warehousing and data quality management.

Oracle makes available application connectors for many Oracle ERP applications such as Oracle E-Business Suite. However, Oracle customers using applications other than those supported by official Oracle OWB connectors have similar requirements.

To help application vendors better serve these customers, we are opening the interfaces used to build OWB connectors for public use.

This whitepaper will:

- Present the business value of building a connector for your own application
- Describe all of the required components of a connector
- Review a real example of a connector created to work with an application not developed by Oracle
- Propose approaches to creating connectors that are not directly accessible by the mechanisms available in OWB

OWB Connectors: Business Opportunities for Oracle Partners

For an application provider, an OWB connector may itself bring enough value to customers to be worth selling on its own. The more complex the application, the greater the need for a logical-level abstraction that simplifies ETL design around the underlying objects.

We anticipate, however, that most applications for which connectors will be built will be used with applications for specific market niches, where there will be other opportunities to add value:

- There may be value in domain-specific data rules. For example, a rule that defines a valid student or course ID number or a domain of legal values for student status (enrolled full time, enrolled part time, on leave, suspended, graduated, etc.) might be useful in an education-related application.
- Your application may benefit from a library of transformations (delivered, for example, as PL/SQL packages) that perform common manipulations of your data as represented by the data returned from the application connector.
- For applications which are frequently used as sources for a data warehouse, a connector simplifies the development of the ETL logic for loading the warehouse and validating the contents of the ODS.
- A complete solution for your application similar to the Data Watch and Repair solution Oracle provides for their customer and product data hubs can add value for applications where master data management requirements apply.

OWB Connectors and OWB and Database Licensing Requirements

Please contact Oracle Warehouse Builder product management for details on licensing requirements for Oracle Warehouse Builder used with third-party application connectors.

INSIDE AN APPLICATION CONNECTOR: CMI XML QUERIES AND CAPABILITIES

As mentioned earlier, the core function of an OWB connector is to expose to the developer only those tables, views, columns, and relationships that they need to work with directly in designing ETL/data integration processes.

The connector uses a series of SQL queries to be executed against your application's metadata to collect information about the objects to expose.

Connectors can support a range of capabilities, depending upon your business requirements and the complexity of the application.

Defining a connector requires that you decide upon the capabilities you will expose and then write the requisite SQL queries. Which queries you define and their complexity depends upon the set of capabilities you choose to implement.

OWB supports a mechanism called *CMI* (Custom Metadata Interface) which is used to register the information about ERP applications with OWB.

The connector developer creates an XML file called a *CMI metadata file* that contains the query definitions and the capability values. (Note that for historical reasons, CMI metadata files shipped with OWB may have the filename extension MIV. This extension is not a requirement; the file can have any name.)

Defining connector capabilities

The connector model in OWB includes a range of capabilities that the connector may implement, including:

- The ability to import metadata for tables, views, and sequences from the application's underlying database schema or from separate metadata tables
- The ability to group imported objects into business areas
- The ability to import metadata about foreign key relationships among tables

For example, some connectors may not expose views at all (or integration with the underlying application may not require them).

When defining a connector, the first step is to decide which of the capabilities you will implement, and then populate the `miv_capabilities` element in the CMI XML file with child elements that specify the required values.

The following table lists the element names for the capabilities, whether the elements are mandatory or optional, and the default value if the element is omitted. Supported values are `true` or `false`. Most of the capabilities are optional and sensible defaults are provided.

Capability Element Name	Description	Required	default
<code>table_supported</code>	Defines whether this connector supports the import of tables.	no	true
<code>view_supported</code>	Whether this connector supports import of information about views.	no	false
<code>sequence_supported</code>	Whether this connector supports import of information about	no	false

	sequences.		
table_name_filter_supported	Whether this connector supports filtering out tables not meant to be exposed based on their names.	no	true
view_name_filter_supported	Whether this connector supports filtering out views not meant to be exposed based on their names.	no	false
sequence_name_filter_supported	Whether this connector supports filtering out sequences not meant to be exposed based on their names.	no	false
business_area_supported	Whether this connector supports organizing imported objects by business area.	no	false
business_area_table_supported	Whether this connector supports organizing imported tables by business area.	no	false
business_area_view_supported	Whether this connector supports organizing imported views by business area.	no	false
business_area_sequence_supported	Whether this connector supports organizing sequences by business area.	no	false
application_owner_supported	Whether this connector exposes objects owned by multiple database-level users (e.g. if each application in a suite stores its objects under a different owner).	no	true
table_fklevel	Whether the connector	no	true

_supported	supports the import of foreign keys.		
reimport_supported	Whether reimport is supported.	no	true
data_object_at_leaf_levels	Whether data objects can be exposed at leaf levels in the tree.	no	true
multiple_tree_supported	Whether a single object can be exposed in multiple places in the tree (useful where the tree includes categories that overlap).	no	true

Defining queries against database and application metadata

A connector must run SQL queries on the underlying database of an application to discover the available tables, views, sequences, and relationships among them, and exposes all of those details to the ETL developer.

The definition of the connector in the CMI XML file includes definitions of the required queries. Each query is wrapped in a specific element in the CMI XML file. Note that some queries are optional, based on the capabilities you support.

The recommended approach to maintaining application metadata for the connector is to set up separate tables that you maintain explicitly that describe all of the metadata. For example, Oracle E-Business Suite and PeopleSoft applications maintain such metadata tables. If you do not maintain separate metadata tables of this sort, then you can query the database dictionary, using WHERE clauses based on naming conventions for tables, views and columns to identify relevant objects to be exposed through the connector. However, this approach can be prone to errors, if, for example, a user creates tables of their own in the underlying schema that conflict with the naming convention.

In the discussion that follows, for the basic queries, a sample query is provided that extracts properly formed metadata for each query from the Oracle database dictionary.

Parameters to Queries

The queries are passed four parameters from OWB:

- **Owner** – the owner of the object against which the query is executed..
- **DbLink** – The name of a database link used to connect to the database containing the application schema
- **Object** – The object against which the query is being executed.

- **Filter** – A condition in the query used to filter out extraneous objects.

Example queries below will illustrate the use of the parameters in a query.

Basic Application Connector Capability

The tables below are shown with reference information about the columns they return, as well as a basic example showing the XML element containing the query. To see how these fit in context of the complete connector file, see the appendix to this document that reproduces a basic connector based on Oracle data dictionary metadata in its entirety.

In the discussion that follows, for the basic queries, a sample query is provided that extracts a reasonable set of metadata from the Oracle database dictionary. In practice, however, we recommend that you maintain a set of separate tables, similar to the E-Business Suite or PeopleSoft metadata tables, which contain such metadata. This approach provides more flexibility with object names and descriptions than relying upon the database data dictionary.

To define a basic connector offering a useful subset of the complete functionality (exposing the available business objects but not grouping them into business areas), use the following steps:

First, in the CMI XML file, specify FALSE for the following capabilities:

- `business_area_supported`
- `business_area_table_supported`
- `business_area_view_supported`
- `business_area_sequence_supported`

Then define the following queries. (Note that some of these queries can return an empty set of rows if appropriate—for example, if you are not using views, can return no rows, or if the `view_supported` capability is false it can be omitted altogether.)

Query MIV_TABLES

Extracts metadata for all tables in the schema to be exposed to the user.

Column name	Description
<code>table_name</code>	Name of the table
<code>business_name</code>	Business name for the table
<code>description</code>	Description of the table

Example:

```
<miv_tables type="SQLStatement" default="true">
  SELECT
    rtrim(table_name) table_name,
    initcap(rtrim(table_name)) business_name,
```

```

        '' description
        FROM all_tables<Parameter name="dblink"/> atc
        WHERE owner=<Parameter name="owner" />
</miv_tables>

```

Query MIV_VIEWS

Extracts metadata for all views in the schema to be exposed to the user.

This element determines how views should be integrated. The context for the query is the schema or schemas selected. The SQL query provided should project the columns in the table below. The query will be invoked for each schema selected, and the parameters will be available within the query for the SQL template provided.

Column name	Description
view_name	Name of the view
business_name	Business name for the view
description	Description of the view

Example:

```

<miv_views type="SQLStatement" default="true">
  SELECT rtrim(v.view_name) view_name,
         Initcap(rtrim(v.view_name)) business_name,
         '' description,
         v.text view_definition
  FROM all_views<Parameter name="dblink"/> v
  WHERE v.owner = <Parameter name = "owner"/>
  ORDER BY v.view_name
</miv_views>

```

Query MIV_SEQUENCES

Extracts metadata for all sequences in the schema to be exposed to the user.

This element determines how sequences should be integrated. The context for the query is the schema or schemas selected. The SQL query provided should project the columns in the table below. The query will be invoked for each schema selected, and the parameters will be available within the query for the SQL template provided.

Column name	Description
view_name	Name of the view
business_name	Business name for the view
description	Description of the view

Example:

```
<miv_sequences type="SQLStatement" default="true">
  SELECT
    rtrim(sequence_name) sequence_name,
    initcap(rtrim(sequence_name)) business_name,
    ' ' description
  FROM all_sequences<Parameter name="dblink"/> s
  WHERE s.sequence_owner = <Parameter name="owner"/>
  ORDER BY sequence_name
</miv_sequences>
```

Query MIV_COLUMNS

Extracts metadata for all columns in all tables in the schema to be exposed to the user. The context for the item is the schema or schemas selected and the table or view. The query provided should project the columns shown in the following table.

Column name	Description
Entity_name	Name of the entity (e.g. table or view)
Column_name	Name of the column
Business_name	Business name of the column
Description	Description of the column
Postion	Integer position of the column
Datatype	Type of the column
Length	Length of the column
Precision	For numeric columns, the precision of the column
Scale	For numeric columns, the scale of the column
Nulls_allowed	Whether the column can have null values

Example:

```
<miv_columns type="SQLStatement" default="true">
```

```

SELECT
  ac.table_name entity_name,
  rtrim(ac.column_name) column_name,
  initcap(rtrim(ac.column_name)) business_name,
  '' description,
  ac.column_id position,
  rtrim(ac.data_type) data_type,
  ac.data_length length,
  ac.data_precision precision,
  ac.data_scale scale,
  ac.nullable isNullable
FROM all_tab_columns<Parameter name="dblink"/> ac
WHERE ac.owner = <Parameter name="owner"/>
ORDER BY ac.table_name, ac.column_id
</miv_columns>

```

Query MIV_UNIQUE_KEYS

Extracts metadata for all unique and primary key constraints in the schema to be exposed to the user.

Column name	Description
Entity_name	Name of the entity
Key_name	Key name
Business_name	Business name
Constraint_type	Type of the constraint. Values: U (unique) or P (primary).
Description	Description of the table

Example:

```

<miv_unique_keys type="SQLStatement" default="true">
  SELECT
    ac.table_name entity_name,
    rtrim(ac.constraint_name) key_name,
    initcap(rtrim(ac.constraint_name)) business_name,
    '' description,
    rtrim(ac.constraint_type) constraint_type
  FROM
    all_constraints ac
  WHERE
    upper(ac.constraint_type) in ('U', 'P')
    AND ac.owner = <Parameter name="owner"/>
  ORDER BY ac.constraint_name
</miv_unique_keys>

```

Query MIV_FOREIGN_KEYS

Extracts metadata for all foreign key relationships.

Column name	Description
-------------	-------------

Entity_name	Name of the entity (typically, table)
Foreign_key_name	Name of the foreign key
BUSINESS_NAME	Business name for the table
DESCRIPTION	Description
unique_key_name	Name of the unique key

Example:

```
<miv_foreign_keys type="SQLStatement" default="true">
SELECT
    ac.table_name entity_name,
    rtrim(ac.constraint_name) foreign_key_name,
    initcap(rtrim(ac.constraint_name)) business_name,
    ' ' description,
    rtrim(ac.r_constraint_name) unique_key_name
FROM all_constraints<Parameter name="dblink"/> ac
WHERE
    upper(ac.constraint_type) = 'R'
    AND ac.owner = <Parameter name="owner"/>
ORDER BY ac.constraint_name
</miv_foreign_keys>
```

Query MIV_KEY_COLUMNS

Extracts metadata for all primary and foreign key in the schema to be exposed to the user.

Column name	Description
Key_name	Name of the key
Column_name	Name of the column storing the key
Position	Integer position of the column in the table

Example:

```
<miv_key_columns type="SQLStatement" default="true">
SELECT
    constraint_name key_name,
    rtrim(substr(column_name, 1, 50)) column_name,
    position
FROM all_cons_columns<Parameter name="dblink"/>
WHERE owner = <Parameter name="owner"/>
</miv_key_columns>
```

Query MIV_FK_TABLES

Extracts metadata for all tables in the schema to be exposed to the user.

Column name	Description
Orig_table_name	Name of the original table

Foreign_table_name	Name of the Foreign table
Foreign_table_description	Description of the foreign table

Example:

```
<miv_fk_tables type="SQLStatement" default="true">
  SELECT rtrim(ac1.table_name) orig_table_name,
         rtrim(ac2.table_name) foreign_table_name,
         substr(atc.comments, 1, 2000)
         foreign_table_description
  FROM
    all_constraints<Parameter name="dblink"/> ac2,
    all_tab_comments<Parameter name="dblink"/> atc,
    all_constraints<Parameter name="dblink"/> ac1
  WHERE upper(ac1.constraint_type) = 'R'
        AND ac1.owner = ac2.owner
        AND ac1.owner = <Parameter name="owner"/>
        AND upper(atc.table_type)='TABLE'
        AND ac2.constraint_name = ac1.r_constraint_name
        AND ac2.table_name = atc.table_name
        AND ac2.owner = atc.owner
  ORDER BY orig_table_name
</miv_fk_tables>
```

Installing the Connector in OWB

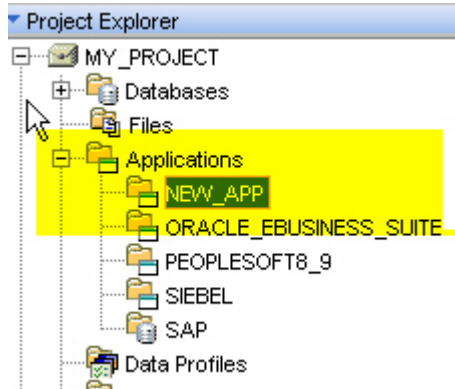
To install your connector in the OWB repository:

1. Copy the XML file with the CMI definitions to a directory accessible on your OWB repository server. For example, on a Windows host you could store the file in C:/CONNECTOR/EXAMPLE_CONNECTOR.XML.
2. Execute the following command to create the application definition:

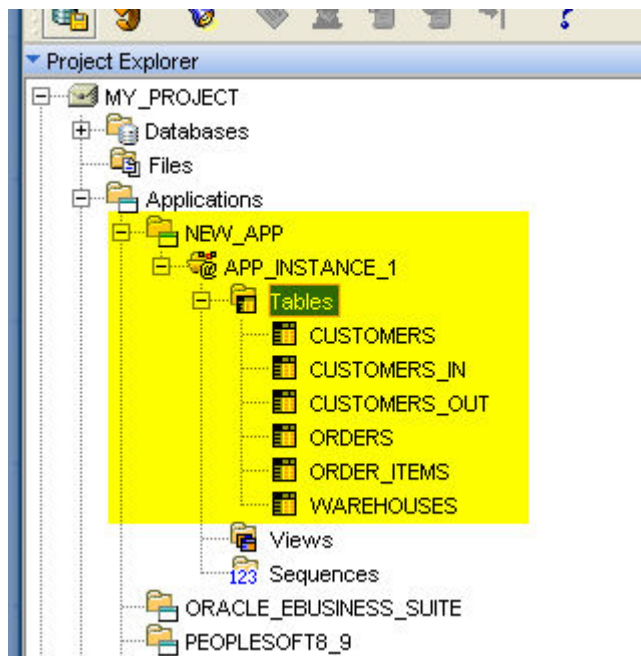
```
OMB+> OMBCREATE CMI_DEFINITION 'NEW_APP' USING
DEFINITION_FILE 'C:/CONNECTOR/EXAMPLE_CONNECTOR.XML';
```

CMI definition NEW_APP created.

The new platform supported by the connector is then visible in the Project Explorer in OWB Design Center, as shown here:



You can then create a module for objects imported from your application, as with any other application connector. See the Oracle Warehouse Builder documentation for details on this process. The final result will look like this:



Uninstalling and Updating a Connector

If you need to modify your connector, for example during development, simply delete all modules that you have created that reference the connector. You can then use the `OMBCREATE CMI_DEFINITION` command to load an updated definition and try the connector again.

Advanced Connector Capability: Supporting Business Areas

Grouping entities into Business Areas

For applications with large and complex schemas representing data associated with different modules, you can group the application objects exposed by the connector into business areas. Business areas can also be nested in a hierarchy for particularly complex applications.

To enable business area support, set the related capability values and define some additional queries as shown in the following text.

For the following example, assume that the application database has metadata tables `business_areas` and `business_area_objects`, which list the business areas, the nesting relations among them, and association of individual objects with business areas. The structure of the tables can be deduced from the queries below.

Business area capability values

The required capability values are as follows:

- Set `business_area_supported` to true.
- Set `business_area_table_supported` to true if tables exposed by the connector should be grouped by business area.
- Set `business_area_view_supported` to true if views exposed by the connector should be grouped by business area.
- Set `business_area_sequence_supported` to true if sequences exposed by the connector should be grouped by business area.

Query MIV_BUSINESS_AREAS

This query determines the business areas that group objects exposed through the connector. The context for the query is the schema or schemas selected.

Column name	Description
<code>business_area_id</code>	ID of the business area.
<code>parent_business_area_id</code>	ID of the parent business area. Should be '0' for each top level business area.
<code>Business_area_name</code>	Name for the business area
<code>Business_name</code>	Name for the top level business area
<code>Description</code>	Description

Example:

```
<miv_business_areas type="SQLStatement" default="true">  
  SELECT business_area_id, parent_business_area_id,
```

```

        business_area_name, business_name, description
    FROM business_areas
    ORDER BY business_area_name;
</miv_business_areas >

```

Query MIV_BUSINESS_AREA_OBJECTS

This query specifies how objects exposed through the connector are grouped under business areas. The context for the item is the schema or schemas selected.

Column name	Description
object_name	Name of the table, view or sequence
parent_business_area_id	ID of the parent business area.
Type	Type of the object. Values can be 'TABLE', 'VIEW', 'SEQUENCE'.
Description	Description of the object.

Example:

```

<miv_business_area_objects
  type="SQLStatement" default="true">
  SELECT object_name, parent_business_area_id,
         type, description
  FROM business_area_objects;
</miv_business_areas>

```

Advanced Connector Capability: Supporting Writeback

Writeback is the ability to use the tables exposed by a connector as targets, rather than just sources.

Tables exposed through the application connector can be used as targets in a mapping. Note, however, that how well this works depends upon your application logic. If it is otherwise safe to directly modify the base tables of your application without going through interfaces that, for example, ensure that bad data is not written, then you can simply use your tables as targets in mappings.

You may, however, want to control updates to your application's master tables. One method of implementing this is:

- Create a parallel set of interface tables alongside the source tables
- Expose those alongside the source tables through the connector metadata
- When creating mappings, only use the interface tables as targets
- Create a process flow that invokes the mapping, then invokes some trusted PL/SQL function that validates the values from the interface tables and then updates the source tables if there are no errors.

A variation on this technique can be used if there is not a direct one-to-one mapping between updates as represented in the interface tables and data as stored

in the base tables. PL/SQL can copy interface table data to multiple base tables, which may not all be exposed through the connector. For example, this technique is used to support writeback through interface tables exposed for the Universal Customer Master customer data hub supported in the Warehouse Builder Data Watch and Repair solution.

Connectors For Sources Without SQL Metadata

As noted, the connector architecture for OWB cannot directly support applications where the metadata cannot be queried using SQL. An example of such an application is Salesforce.com. This limitation may be addressed in a future release of OWB. In the interim, you can use solutions such as ODBC drivers for such sources to enable query their metadata with SQL.

SUMMARY

The chief business benefit of using OWB Connectors is improved developer productivity in any data integration, data warehousing or data quality project where the source or target schemas are more complex than the system of logical-level business objects represented by the application.

Constructing a connector reduces the effort required to bring the full capabilities enabled by OWB and the Oracle database to bear on the data from the application: data quality, integration with other applications and data sources/targets, data warehousing, data mining and business intelligence.

The process of constructing a connector and adding it to OWB is straightforward. For application providers, building OWB application connectors can improve the value of the application to the customer, and selling connectors, possibly with supporting content such as data rules, creates new business opportunities around the Oracle Database/OWB ecosystem.

Where application providers do not supply OWB connectors themselves, customers or third parties with access to the metadata for the application's schema can build them for their own use if needed.

Oracle Warehouse Builder product management welcomes contacts from any application connector developers.

APPENDIX: COMPLETE BASIC CONNECTOR XML LISTING

```
<?xml version="1.0"?>
<miv>
<miv_capabilities type="ResultSet">
  <table_supported>true</table_supported>
  <view_supported>true</view_supported>
  <sequence_supported>true</sequence_supported>
  <table_name_filter_supported>
    true
  </table_name_filter_supported>
  <view_name_filter_supported>
    true
  </view_name_filter_supported>
  <sequence_name_filter_supported>
    true
  </sequence_name_filter_supported>
  <business_area_supported>
    false
  </business_area_supported>
  <business_area_table_supported>
    false
  </business_area_table_supported>
  <business_area_view_supported>
    false
  </business_area_view_supported>
  <business_area_sequence_supported>
    false
  </business_area_sequence_supported>
  <application_owner_supported>
    true
  </application_owner_supported>
  <table_fklevel_supported>
    true
  </table_fklevel_supported>
  <reimport_supported>
    true
  </reimport_supported>
  <data_object_at_leaf_levels>
    false
  </data_object_at_leaf_levels>
  <multiple_tree_supported>
    false
  </multiple_tree_supported>
</miv_capabilities>
<miv_tables type="SQLStatement" default="true">
  SELECT
    rtrim(table_name) table_name,
    initcap(rtrim(table_name)) business_name,
    '' description
  FROM all_tables<Parameter name="dblink"/> atc
  WHERE owner=<Parameter name="owner" />
</miv_tables>

<miv_views type="SQLStatement" default="true">
  SELECT rtrim(v.view_name) view_name,
    Initcap(rtrim(v.view_name)) business_name,
    '' description,
```

```

        v.text view_definition
    FROM all_views<Parameter name="dblink"/> v
    WHERE v.owner = <Parameter name = "owner"/>
    ORDER BY v.view_name
</miv_views>

<miv_sequences type="SQLStatement" default="true">
    SELECT
        rtrim(sequence_name) sequence_name,
        initcap(rtrim(sequence_name)) business_name,
        '' description
    FROM all_sequences<Parameter name="dblink"/> s
    WHERE s.sequence_owner = <Parameter name="owner"/>
    ORDER BY sequence_name
</miv_sequences>

<miv_columns type="SQLStatement" default="true">
    SELECT
        ac.table_name entity_name,
        rtrim(ac.column_name) column_name,
        initcap(rtrim(ac.column_name)) business_name,
        '' description, ac.column_id position,
        rtrim(ac.data_type) data_type,
        ac.data_length length,
        ac.data_precision precision,
        ac.data_scale scale,
        ac.nullable isNullable
    FROM all_tab_columns<Parameter name="dblink"/> ac
    WHERE ac.owner = <Parameter name="owner"/>
    ORDER BY ac.table_name, ac.column_id
</miv_columns>

<miv_unique_keys type="SQLStatement" default="true">
    SELECT ac.table_name entity_name,
        rtrim(ac.constraint_name) key_name,
        initcap(rtrim(ac.constraint_name)) business_name,
        '' description,
        rtrim(ac.constraint_type) constraint_type
    FROM all_constraints<Parameter name="dblink"/> ac
    WHERE upper(ac.constraint_type) in ('U', 'P')
    AND ac.owner = <Parameter name="owner"/>
    ORDER BY ac.constraint_name
</miv_unique_keys>

<miv_foreign_keys type="SQLStatement" default="true">
    SELECT
        ac.table_name entity_name,
        rtrim(ac.constraint_name) foreign_key_name,
        initcap(rtrim(ac.constraint_name)) business_name,
        '' description,
        rtrim(ac.r_constraint_name) unique_key_name
    FROM all_constraints<Parameter name="dblink"/> ac
    WHERE upper(ac.constraint_type) = 'R'
    AND ac.owner = <Parameter name="owner"/>
    ORDER BY ac.constraint_name
</miv_foreign_keys>

```

```

<miv_key_columns type="SQLStatement" default="true">
  SELECT constraint_name key_name,
  rtrim(substr(column_name, 1, 50)) column_name, position
  FROM all_cons_columns<Parameter name="dblink"/>
  WHERE owner = <Parameter name="owner"/>
</miv_key_columns>

<miv_fk_tables type="SQLStatement" default="true">
  SELECT
    rtrim(ac1.table_name) orig_table_name,
    rtrim(ac2.table_name) foreign_table_name,
    substr(atc.comments, 1, 2000)
      foreign_table_description
  FROM
    all_constraints<Parameter name="dblink"/> ac2,
    all_tab_comments<Parameter name="dblink"/> atc,
    all_constraints<Parameter name="dblink"/> ac1
  WHERE upper(ac1.constraint_type) = 'R'
    AND ac1.owner = ac2.owner
    AND ac1.owner = <Parameter name="owner"/>
    AND upper(atc.table_type)='TABLE'
    AND ac2.constraint_name = ac1.r_constraint_name
    AND ac2.table_name = atc.table_name
    AND ac2.owner = atc.owner
  ORDER BY orig_table_name
</miv_fk_tables>

</miv>

```



Extending OWB: Building New Application Connectors

December 2008

Author: Antonio Romero

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2008, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle, JD Edwards, and PeopleSoft, are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.