Oracle Warehouse Builder 11.2: Platform and Application Adapter Extensibility Cookbook
Introduction

This paper provides an overview of Oracle Warehouse Builder (OWB) 11gR2 extensibility to support non-Oracle data sources. It describes the components you must construct to define the data types for a new platform and methods of metadata extraction from that platform.

Previous versions of OWB 11gR2 only supported the Oracle Database as a platform, with non-Oracle databases accessed through gateways that masked their native data types. Reverse-engineering of metadata from non-Oracle data sources was possible only through gateways.

The new release overcomes these limitations with a more flexible type system and enhancements to the OWB Custom Metadata Interface (CMI) that supports hot-pluggable reverse engineering capabilities. Combined with JDBC connectivity and code template mapping technology from Oracle Data Integrator, these features enable the use of non-Oracle data sources in OWB mappings.

Examples of the components described in this paper will be available on the Warehouse Builder SDK Home Page, here:

http://www.oracle.com/technology/products/warehouse/SDK
OWB 11gR2: New Platforms for New Possibilities

The most significant architectural changes to Oracle Warehouse Builder 11gR2 compared to previous releases are in the area of native support for non-Oracle databases. These involved augmenting many Oracle Database-specific elements of the architecture with more general, extensible and platform-neutral implementations. The extension mechanisms are intended to be public and can be used by developers to add support for non-Oracle platforms, including relational databases and many other data sources.

The most common use case for these capabilities is adding support for new relational databases. However, these capabilities also facilitate using Warehouse Builder in less conventional data integration scenarios, by enabling a relational presentation of other data sources within ETL mappings:

- A multidimensional database, such as Essbase
- Cloud-based services that support remote queries, such as Twitter or Facebook
- Cloud-based applications, such as Salesforce.com
- An LDAP server or a mail server
- Collections of unstructured data such as PDF or HTML documents

Customers are already using OWB with these techniques for many of these applications today.

Architectural Changes for Extensibility in OWB 11gR2

The following diagram summarizes the architectural changes in OWB 11gR2 related to supporting new platforms.
The new elements include:

- An extensible platform framework, for representing native data types from heterogeneous databases
- Extensions to the mapping model to separate mapping logic from code generation language and target platform
- Support for diverse connectivity technologies beyond gateways and ODBC
- A new code template-based code generator, for generating native EL-T code in SQL and other languages, to run on heterogeneous platforms
- Extensions to the metadata extraction framework to support non-Oracle SQL and Java for new data source and application types in the OWB Custom Metadata Interface (CMI)

From the standpoint of a developer adding a new database (or database-like source) to OWB, the concerns are:

- Selecting a connectivity/data movement technology
- Defining a platform
- Extracting metadata
- Generating code

In practice, connectivity is often JDBC for non-Oracle databases, and in such cases, code generation can be handled with generic JDBC templates.
Platform Definitions

A platform definition provides description of all the data types for the platform, and how to map them to OWB’s internal type system of generic types. In the case of Oracle and sources accessed through gateways, these are Oracle’s native data types; for other sources accessed natively, you define these types yourself. OWB uses these definitions to translate different data types among platforms, in ETL mappings that move data across platforms.

Metadata Reverse-Engineering: Custom Metadata Interface (CMI)

For a new platform, you must define for OWB how to reverse-engineer, or extract metadata from, a data source about the objects defined there—tables, views, columns, sequences, primary keys, and foreign key relationships.

For Oracle sources and sources accessed through gateways, OWB uses standard queries against the database data dictionary to extract the needed metadata.

For sources accessed through JDBC, OWB can use analogous queries and calls to the JDBC driver to extract basic metadata about the source.

In some cases, however, it can be useful or necessary to define custom metadata extraction methods for a given platform. For such cases, OWB provides the Custom Metadata Interface (CMI), an open API used for registering new metadata extraction methods.

A CMI definition allows you to describe custom metadata extraction methods for a given platform, which can take the form of custom (Oracle or native SQL) queries or, for non-SQL platforms, references to a Java class that extracts the metadata using whatever means the platform provides.

A CMI definition file (also called a MIV file for historical reasons) is an XML file containing the required information for the source. Once a platform is defined, you can associate a CMI with it.

Custom Metadata Interface and Application Adapters

OWB application adapters (previously called application connectors) are also built using CMI metadata definitions. They add an additional layer of metadata about ERP and similar application data sources which are in turn hosted on databases accessed through Oracle native connectivity, gateways and JDBC. They make possible metadata extraction from application-specific metadata catalogs, presenting the ETL developer with an application logic-level view of only relevant tables, views and sequences, and presenting these objects organized into business areas.

For a developer building a new application adapter, the two concerns are:

• Defining support for the database platform hosting the application;

• Defining queries to get metadata for the application, given the platform.
This paper covers CMI definitions for applications as well as database platforms, because they are quite similar in construction to CMI definitions for new platforms. Details on registering a new application CMI are provided in a later section of this paper.

Data Extraction, Movement and Integration

You must also provide the means for OWB to generate code that moves extracts, loads, and integrates data on the new platform.

For Oracle database sources and gateway sources, data movement uses database links. OWB can generate native Oracle SQL and PL/SQL code for these directly.

For non-Oracle data sources, code templates are used for code generation, with metadata from the source or target and the mapping are used to populate the templates. OWB 11gR2 comes with templates for Oracle, DB2, SQL Server, generic JDBC, XML and some others out of the box. More templates can be added or created as needed.

Where JDBC support is not available, or where faster performance is needed, code templates can describe steps needed for other methods, such as database-specific SQL or even bulk data extraction and loading through flat files.

Creating optimal code templates specific to a new platform is an important topic, but is beyond the scope of this paper.

Comparing OWB Extensibility and Similar ODI Features

OWB’s platform support can be understood as analogous to Oracle Data Integrator (ODI)’s technology.

- OWB’s platforms are conceptually similar to ODI technologies; both include metadata about database or other data source/target platforms, and OWB platforms can be created based on ODI technologies.

- OWB’s metadata extraction capabilities play roughly the same role as ODI’s reverse-engineering knowledge modules (RKMs), though the underlying implementations and the means by which they are exposed to the user are quite different.

- OWB code templates are essentially identical to ODI knowledge modules. (Because the term “module” already referred to another concept in the OWB user interface, it was decided to rename knowledge modules in OWB.) As a result, in general the same data extraction, movement, loading, CDC and transformation are available in OWB as in ODI.
OWB Platform Definitions

The OWB type system in OWB 11gR2 supports the definition of new platforms with handling of the native data types for these platforms. A set of platform-neutral core data types, sometimes referred to as the generic data type hub, is used internally. A platform definition describes the native types for a new platform and maps them to and from the generic data types. When operators table, view, sequence, and function operators are used in ETL mappings, the platform where the underlying bound object is defined provides the metadata for the object’s column types.

Pre-seeded platforms, ready for use out of the box, include Oracle, File (for flat files), DB2 UDB and SQL Server. Users can create more at any time. (Other database types presented in Location Explorer, such as Sybase, Informix and Teradata, are present because OWB can connect to these through gateways, which does not require a platform definition. You can define native platforms for those databases using the techniques in this paper, if you want to access them without gateways, but those are not part of OWB 11gR2 as shipped.)

Creating a New Platform Definition

Platform definitions are managed through the OMB*Plus scripting language. There are three required parts to a platform definition:

- Defining a platform object and setting its properties
- Setting the platform types for the platform and their properties
- Specifying the type mappings for each platform type to and from a generic type

Defining a Platform Object

To define a platform, use an OMBCREATE PLATFORM command to create the platform. For example:

```
OMBCREATE PLATFORM 'MYSQL'
```

Then use a series of OMBALTER PLATFORM commands to set its properties.

```
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (prop-name) VALUES (prop-value)
```

The properties you need to define are summarized in the following table. For more details, see Oracle Warehouse Builder Sources, Targets and Connectivity Guide, chapter 11, “Creating New Platforms.”

<table>
<thead>
<tr>
<th>PROPERTY</th>
<th>REQUIRED</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSINESS_NAME</td>
<td>No</td>
<td>A business name to display for the platform. Can be mixed case, include white space etc.</td>
</tr>
<tr>
<td>PROPERTY</td>
<td>REQUIRED</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DRIVER_CLASS</td>
<td>Yes</td>
<td>The Java class name for the JDBC driver to use with this platform. Class must exist in the JAR file for the driver. Examples: ‘com.mysql.jdbc.Driver’ (for MySQL) ‘com.sunopsis.jdbc.driver.xml.SnpsXmlDriver (for the XML driver from ODI)</td>
</tr>
<tr>
<td>URI_TEMPLATE</td>
<td>Yes</td>
<td>The template for the JDBC URI that identifies the target database. Example: jdbc:mysql://&lt;host&gt;[:&lt;port&gt;]/[&lt;database&gt;][?&lt;property&gt;=&lt;value&gt;][&amp;&lt;property&gt;=&lt;value&gt;...]]</td>
</tr>
<tr>
<td>COL_ALIAS_WORD</td>
<td>No</td>
<td>The word used to identify a column alias in a SQL SELECT clause in the target SQL dialect. Example: ‘as’ Note: Not all databases support or require such a string.</td>
</tr>
<tr>
<td>TAB_ALIAS_WORD</td>
<td>No</td>
<td>The word used to identify a table alias in a SQL SELECT clause in the target SQL dialect. Example: ‘as’ Note: Not all databases support or require such a string.</td>
</tr>
<tr>
<td>DATE_MASK</td>
<td>Yes</td>
<td>The text string describing the syntax for specifying a date in the target SQL dialect, with any delimiters etc. required. Example: ‘datetime’</td>
</tr>
<tr>
<td>DATE_FCT</td>
<td>Yes</td>
<td>The text string for a function call that returns the current date. Example: ‘CURRENT_DATE()’ for MySQL, ‘sysdate’ for Oracle. Note the inclusion of parentheses in the MySQL case.</td>
</tr>
<tr>
<td>DDLNULL</td>
<td>Yes</td>
<td>The word used to describe a column that can contain empty values (column known as NULLABLE).</td>
</tr>
<tr>
<td>NUMERIC_MASK</td>
<td>Yes</td>
<td>The text string describing the syntax for specifying a numeric value in the target SQL dialect, with any delimiters etc.</td>
</tr>
<tr>
<td>DEFAULT_MAX_NAME_LEN</td>
<td>Yes</td>
<td>The maximum length of a datastore (table) name in number of characters. All table names generated by the product are truncated to this size.</td>
</tr>
<tr>
<td>DEFAULT_NAME_LEN_SEMANTICS</td>
<td>No</td>
<td>Name length semantics (bytes or characters) – Default: characters.</td>
</tr>
<tr>
<td>SPECIAL_MAX_NAME_LEN</td>
<td>No</td>
<td>Name lengths for the object’s associated second-class-objects, that is, objects such as indexes, columns, and so on associated with a table. For example: INDEX=18:COLUMN=30:PACKAGE=18</td>
</tr>
<tr>
<td>SPECIAL_NAME_LEN_SEMANTICS</td>
<td>No</td>
<td>Name length semantics (bytes or characters)</td>
</tr>
<tr>
<td>ESCAPE_CHAR</td>
<td>No</td>
<td>The escape character for special characters (e.g. for Oracle, a double-quote ”)</td>
</tr>
<tr>
<td>ENCLOSURE_CHAR</td>
<td>No</td>
<td>The closure character (for Oracle, a backslash )</td>
</tr>
<tr>
<td>RESERVED_WORDS</td>
<td>No</td>
<td>A comma-separated list of the platform’s reserved words that require quoting.</td>
</tr>
<tr>
<td>ILLEGAL_CHARS</td>
<td>No</td>
<td>Illegal characters for names in this platform. For example, -1@#$%/&amp;‘()+' on DB2. Note the use of XML entities for some characters.</td>
</tr>
<tr>
<td>ILLEGAL_LEADING_CHARS</td>
<td>No</td>
<td>Illegal leading for names in this platform. For example, 1234567890-1@#$%/&amp;‘()+' on DB2. Note the use of XML entities for some characters.</td>
</tr>
<tr>
<td>DEFAULT_IMPORTER</td>
<td>No</td>
<td>Specifies the custom CMI definition XML file if any that provides queries and/or Java classes required for reverse-engineering.</td>
</tr>
<tr>
<td>VARCHAR_MASK</td>
<td>No</td>
<td>Runtime properties used by the Journaling KMs in creating journaling tables.</td>
</tr>
</tbody>
</table>
### PROPERTY | REQUIRED | DESCRIPTION
--- | --- | ---
LOCAL_OBJECT_MASK | Yes | Syntax for naming objects locally. Placeholders are available as follows:
- `%CATALOG` to represent the physical catalog name
- `%SCHEMA` to represent the schema name
- `%OBJECT` to represent the object name

REMOTE_OBJECT_MASK | Yes | Syntax for naming remote objects from the target platform. Placeholders are available as follows:
- `%CATALOG` to represent the physical catalog name
- `%SCHEMA` to represent the schema name
- `%OBJECT` to represent the object name
- `%DSERVER` to represent the data server
Assign an empty string if the database type does not support naming remote objects.

For example, the following OMB*Plus commands create a platform object for MySQL and set the needed properties:

```ombplus
OMBCREATE PLATFORM 'MYSQL' SET PROPERTIES (BUSINESS_NAME) VALUES ('MySQL')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (DRIVER_CLASS) VALUES ('com.mysql.jdbc.Driver')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (URL_TEMPLATE) VALUES ("jdbc:mysql://<host>[:<port>]\/<database>\[?<property>=<value>\[&<property>=<value>\...\]]\")
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (COL_ALIAS_WORD) VALUES ('as')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (TAB_ALIAS_WORD) VALUES ('as')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (DATE_MASK) VALUES ('datetime')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (DATE_FCT) VALUES ('CURRENT_DATE()')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (DDLNULL) VALUES ('')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (VARCHAR_MASK) VALUES ('varchar(%L)')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (LOCAL_OBJECT_MASK) VALUES ('%CATALOG.%OBJECT')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (DEFAULT_MAX_NAME_LEN) VALUES ('30')
OMBALTER PLATFORM 'MYSQL' SET PROPERTIES (REMOTE_OBJECT_MASK) VALUES ('')
```

### Defining Platform Types

Once the platform is defined, create and set properties for the individual platform types associated with it. The command for adding a platform type to a platform is:

```ombplus
OMBALTER PLATFORM 'platform_name' ADD PLATFORM_TYPE 'type_name'
```

where `type_name` is the name of a data type on the target platform. To set properties on the platform type, the command is:

```ombplus
OMBALTER PLATFORM 'platform_name' MODIFY PLATFORM_TYPE 'type_name' SET PROPERTIES(name1,...) VALUES ('value',...)
```
The platform type properties to set are listed in the following table.

<table>
<thead>
<tr>
<th>PROPERTY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHYSICAL_NAME</td>
<td>Physical name.</td>
</tr>
<tr>
<td>BUSINESS_NAME</td>
<td>A business name to display for the platform. Can be mixed case, include white space etc. Examples: “SQL Server”, “MySQL”. If none provided, the platform name will be used.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Description of the data type.</td>
</tr>
<tr>
<td>ICON</td>
<td>Icon used for the data type. ? how ??</td>
</tr>
<tr>
<td>IS_DEFAULT</td>
<td>Boolean – ‘true’ if this is the default type for the platform, ‘false’ otherwise</td>
</tr>
<tr>
<td>NOTE</td>
<td>A note.</td>
</tr>
<tr>
<td>SYNTAX</td>
<td>Syntax for DDL in the target language. Can include placeholders %size, %precision and %scale.</td>
</tr>
</tbody>
</table>

P1, P1MIN, P1MAX, P1DEFAULT, P1TYPE and P2, P2MIN, P2MAX, P2DEFAULT, P2TYPE

Additional properties appropriate to the platform type, along with minimum, maximum, default and type information.

P1 and P2 values can be:
- ‘size’ for length
- ‘precision’ and ‘scale’ for numeric types

P2MIN and P2MAX are minimum and maximum values
P1DEFAULT and P2DEFAULT are default values for P1 and P2
P1TYPE and P2TYPE can be either ‘range’ (in which case P1MIN and P1MAX are required) or left blank.

For example, for a numeric type on MySQL, P1, P1MIN, P1MAX, P1DEFAULT and P1TYPE are ‘precision’, ‘1’, ‘1000’, ‘1’ and ‘range’ to indicate that property P1 ‘precision’ can range from 1 to 1000 and defaults to 1. P2 can specify ‘scale’, ranging from ‘0’ (P2MIN) to ‘18’ (P2MAX) and defaulting to ‘0’ (P2TYPE).

Defining Platform Type Maps to and from Generic Types

For each platform type, you must define how to map it to and from the generic types used by OWB. A TO_PLATFORM_TYPEMAP specifies how to map from generic to platform-specific, and a FROM_PLATFORM_TYPEMAP specifies how to map from platform-specific to generic.

The command for mapping platform types to generic types is:

```sql
OMBALTER PLATFORM 'PLATFORM' ADD FROM_PLATFORM_TYPEMAP 'typemapname' 
    SET PROPERTIES (FROM_DATATYPE, TO_DATATYPE) VALUES ('typename', 'gtypename')
```

where `typename` is a platform type name you have defined, and `gtypename` is one of the generic type names, listed in the table below:
The full list of generic data types and their valid lengths, ranges and so on are described in the Oracle Warehouse Builder Sources, Targets and Connectivity Guide, chapter 11, “Creating New Platforms.” You can also find there examples of how the data types for DB2 and SQL Server are mapped to the generic data types. Use these examples as guidelines when defining the data types and type mappings for your own platforms.

The command for mapping generic types to platform types is:

```
OMBALTER PLATFORM 'PLATFORM' ADD TO_PLATFORM_TYPEMAP 'typemapname' 
  SET PROPERTIES [FROM_DATATYPE, TO_DATATYPE] VALUES ('typename', 'gtypename')
```

where typename is a platform type name you have defined, and gtypename is one of the generic type names.

The full list of platform type map properties you can set is defined in the following table.

<table>
<thead>
<tr>
<th>PROPERTY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>The physical name of the type map.</td>
</tr>
<tr>
<td>BUSINESS_NAME</td>
<td>Business name of the type map.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Description of the type map.</td>
</tr>
<tr>
<td>FROM_DATATYPE</td>
<td>The data type to map from.</td>
</tr>
<tr>
<td>TO_DATATYPE</td>
<td>The data type to map to.</td>
</tr>
<tr>
<td>CONDITION1</td>
<td>Used for defining the criteria on when this data type mapping is valid—for example, testing a</td>
</tr>
<tr>
<td>CONDITION2</td>
<td></td>
</tr>
</tbody>
</table>
For more details see *Oracle Warehouse Builder Sources, Targets and Connectivity Guide*, chapter 11, “Creating New Platforms.”

**Platform Definitions: Putting it All Together**

All the commands required for the MySQL platform are, for this example, in a tcl script `platform_mysql.tcl`. You can run this script from the OMB*Plus window in Design Center, as shown here:

```
OMB*Plus View

OMB*Plus: Release 11.2.0.1.0
Copyright (c) 2000, 2009, Oracle. All rights reserved.
OMB*Plus source c:\owb_mysql\platform_mysql.tcl
Platform MYSQL created.
Platform MYSQL altered.
Platform MYSQL altered.
Platform MYSQL altered.
Platform MYSQL altered.
Platform MYSQL altered.
Platform MYSQL altered.
```

Once you define a new platform, the platform appears in the list of database types in Project Explorer in OWB Design Center, alongside the seeded databases, and will be added to Location Explorer as well.
OWB Platform Utilities: Managing Platform Definitions

Creating a platform definition from scratch through scripting can be quite laborious. A useful (but unsupported) package of utilities, the OWB Platform Assistants, saves time by helping you create a good first version of a platform definition, based on one of two sources:

- The JDBC driver for a data source can provide much of the metadata OWB needs for a platform definition, through methods defined in the interface java.sql.DatabaseMetadata such as getTypeInfo.
- ODI Technologies serve the same purpose as OWB platform definitions, and contain substantially similar metadata, but are stored as XML files using a different syntax. The Platform Generator utility can import ODI technologies and create corresponding platform definitions.

Another expert in the package provides a GUI view of the platform definitions in your workspace, something not otherwise available in Design Center.

Installing the OWB Platform Utilities

The platform utilities are delivered in a Zip file from OTN. The Zip file contains:

- platform_assistants.mdl, which defines the three experts that make up the utility
platformutil.tcl, a helper script used by the experts.

Unzip this file into the Oracle Home containing your OWB installation, then import the mdl file into your workspace. The following experts will be created in the Public Experts folder:

- CREATE_PLATFORM_FROM_JDBC – Creates a platform from a JDBC driver
- CREATE_PLATFORM_FROM_TECH – Creates a platform from an ODI Technology definition.
- VIEW_PLATFORM – Displays the definition of a platform.

**VIEW_PLATFORM: Viewing Platform Definitions**

To view a platform definition:

- In the Public Experts folder, under PLATFORM_ASSISTANTS, right-click VIEW_PLATFORM. A list of platforms is displayed.

For any platform, you can view the platform’s properties, defined types, and mappings to and from OWB’s generic types.

When you are done viewing platform definitions, click “Cancel” to exit the expert.

**Generating a Platform Definition from JDBC Driver Metadata**

Ensure that the JDBC driver you need is installed in $OWB_HOME/lib/ext, so that OWB can use it. (As with any JAR file added in lib/ext, if you install a new one you must restart Design Center for OWB to recognize and use it.) Also ensure that you have a target system for the driver to connect to.
Then, start the CREATE_PLATFORM_FROM_JDBC expert. You will be prompted for the following information:

- the platform name and business name
- connection details including: the JDBC driver class name, the url to connect through the driver, and a username and password for connecting.

Enter valid responses, and OWB will connect to the target and define a platform object for you. It will generate the types for the platform, and any default mappings it can determine between the newly created platform and the hub types. (If unable to connect, platform definition fails and you must run the expert again.) If you look in the locations tree, you will see a new platform with your specified name (or business name) and you can create new JDBC locations in the usual way. You may need to customize the platform further, but this is a starting point.

Generating a Platform Definition from an ODI Technology

To generate a platform definition from an ODI technology file, run the CREATE_PLATFORM_FROM_TECH expert. You will be prompted for a technology file, the class of the JDBC driver for the platform (which must be installed), and the template URL for connecting with the driver.
When you click OK, the expert will create platform, types, and type mappings for you.

Extracting Metadata: Custom Metadata Interface Definitions

In addition to defining a platform, you need to be able to extract metadata from sources implementing that platform: tables, views, sequences, primary and foreign key relationships, and so on. Custom Metadata Interface (CMI) definition enables you to define custom metadata extraction procedures for data sources, including both databases and ERP applications hosted in supported databases.

A CMI definition describes the following:

- Metadata extraction capabilities to be exposed in OWB
- For platforms where SQL queries can be used to extract metadata, the queries that return the required metadata
- For platforms where SQL queries are not supported, the name of a Java class that implements the metadata extraction procedures

A CMI definition is stored in an XML file called a CMI definition file.

Custom Metadata Extraction Methods: Oracle, JDBC and Java

In general, for Oracle database sources or sources accessed through gateways or JDBC, a CMI definition is not needed. OWB can query these sources out of the box and return metadata about basic objects: tables, views, sequences and so on. For Oracle sources and gateway sources, by default OWB queries the Oracle data dictionary. For JDBC sources, by default OWB uses metadata provided by the JDBC driver.

In either case, however, you may want to override this default metadata extraction behavior. Your options are:

- For an Oracle (or gateway) data source, you can define alternative Oracle SQL queries in the CMI definition file.
- For a JDBC source, you can define alternative queries in the CMI definition file specific to the SQL dialect of the source.
- When working with a data source that does not support SQL metadata extraction at all (for example, a cloud-based ERP or other SAAS application), you can implement a Java-based custom metadata extractor that implements a public OWB interface (part of the OWB API), and returns metadata for the source in a well-defined format.

(Note: Where SQL is not supported for metadata extraction, it will typically not be supported for data extraction either. You will generally have to use a custom load code template/LKM for
data extraction from the source and loading into a database target. Refer to the Warehouse Builder 11gR2 and Oracle Data Integrator 10.1 documentation for details on writing code templates.)

Uses of CMI for Databases and Applications

CMI definitions can be used to describe metadata extraction processes for database sources (which appear in the location tree as new database location types) or for ERP application sources (which appear in the Project tree as new Applications types).

Reasons to add CMIs include:

- For JDBC database sources, you may want to customize the queries used based on details of the platform or limitations of the metadata returned from the JDBC driver.

- For application data sources, there are often application-level catalogs that should be queried instead of the underlying database metadata. For example, support for E-Business Suite, Peoplesoft and Siebel in OWB is based on CMI definitions that query application catalogs for metadata and group returned objects into business areas. Even where there are no separate catalogs, you can use custom queries to return a subset of objects, excluding application-internal objects that the ETL developer should not manipulate directly.

- For any database source with a large and complex set of objects, you may want to limit the metadata returned for developers’ convenience.

- When working with complex data sources that have large numbers of tables, views, or columns, some of which may not be useful to ETL developers (or even safe to manipulate directly), you can define queries in the CMI definition that limit the objects returned to those that the ETL developer needs.

CMI Definitions and Business Areas

A basic CMI can be used to import the following information:

- Columns
- Tables/views/sequences
- Constraints
- Functions
- Schemas

Additionally, a CMI definition can group objects returned into business areas, so that the user can more easily manage large numbers of objects by category or filter objects during metadata import. Business areas can be useful when working with either application-level catalogs or very complex databases.
Note that Java-based CMI do not support business area functionality.

Contents of a CMI/MIV Definition XML File
The XML in a CMI definition file defines the capabilities of the CMI extractor, and the queries or Java API calls that actually extract the metadata. An example will show the overall structure of a CMI definition XML document:

```xml
<?xml version="1.0"?>
<miv>
  <miv_capabilities type="ResultSet">
    <table_supported>true</table_supported>
    <view_supported>true</view_supported>
    <sequence_supported>true</sequence_supported>
    ...more capabilities...
  </miv_capabilities>

  <miv_tables type="SQLStatement" default="true">
    ...query...
  </miv_tables>

  <miv_views type="SQLStatement" default="true">
    ...query...
  </miv_views>

  <miv_sequences type="SQLStatement" default="true">
    ...query...
  </miv_sequences>

  ... more elements with queries...
</miv>
```

The top level element is always the miv element. Details of the miv_capabilities and query elements will be explained in the following sections.

CMI Capabilities Definition
The CMI model in OWB includes a range of capabilities that the metadata extractor can 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 import metadata about foreign key relationships among tables
- The ability to group imported objects into business areas

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 `mi_v_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 and the default capability value if the element is omitted. Supported values are true or false. Most of the capabilities are optional and sensible defaults are provided.

<table>
<thead>
<tr>
<th>CAPABILITY ELEMENT NAME</th>
<th>DESCRIPTION</th>
<th>REQUIRED</th>
<th>DEFAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_supported</td>
<td>Whether table metadata can be imported</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>view_supported</td>
<td>Whether view metadata can be imported</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>sequence_supported</td>
<td>Whether sequence metadata can be imported</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>table_name_filter_supported</td>
<td>Whether tables can be filtered by name during import</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>view_name_filter_supported</td>
<td>Whether views can be filtered by name during import</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>sequence_name_filter_supported</td>
<td>Whether sequences can be filtered by name during import.</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>business_area_supported</td>
<td>Whether this CMI supports organizing imported objects by business area.</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>business_area_table_supported</td>
<td>Whether this CMI supports organizing imported tables by business area.</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>business_area_view_supported</td>
<td>Whether this CMI supports organizing imported views by business area.</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>business_area_sequence_supported</td>
<td>Whether this CMI supports organizing sequences by business area.</td>
<td>no</td>
<td>false</td>
</tr>
<tr>
<td>application_owner_supported</td>
<td>Whether this CMI exposes objects owned by multiple database-level users (e.g. if each application in a suite stores its objects under a different owner).</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>table_fklevel_supported</td>
<td>Whether the connector supports the import of foreign keys.</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>reimport_supported</td>
<td>Whether metadata reimport is supported.</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>data_object_at_leaf_levels</td>
<td>Whether data objects can be exposed at leaf levels in the tree.</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>multiple_tree_supported</td>
<td>Whether a single object can be exposed in multiple places in the tree (useful where business areas are not mutually exclusive, e.g. “Customer tables” vs. “Web storefront application tables”).</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>CAPABILITY ELEMENT NAME</td>
<td>DESCRIPTION</td>
<td>REQUIRED</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------------------------------------------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>function_supported</td>
<td>Whether function metadata can be imported</td>
<td>no</td>
<td>true</td>
</tr>
<tr>
<td>function_name_filter_supported</td>
<td>Whether functions can be filtered by name during import</td>
<td>no</td>
<td>true</td>
</tr>
</tbody>
</table>

The subsequent elements miv_tables, miv_views, miv_sequences and so on define the queries to extract the required metadata from the source.

A complete reference for the queries is included in the appendix to this document.

**CMI Query Definitions and Query Parameters**

The query elements you can define are listed in the following table:

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>miv_tables</td>
<td>Query for extracting table metadata</td>
</tr>
<tr>
<td>miv_views</td>
<td>Query for extracting view metadata</td>
</tr>
<tr>
<td>miv_sequences</td>
<td>Query for extracting sequence metadata</td>
</tr>
<tr>
<td>miv_columns</td>
<td>Query for extracting column metadata</td>
</tr>
<tr>
<td>miv_unique_keys</td>
<td>Query for extracting unique keys</td>
</tr>
<tr>
<td>miv_foreign_keys</td>
<td>Extracts metadata for foreign key relationship</td>
</tr>
<tr>
<td>miv_key_columns</td>
<td>Extracts metadata for all primary and foreign key columns in the schema to expose to the user</td>
</tr>
<tr>
<td>miv_fk_tables</td>
<td>Extracts metadata for all relationships between primary and foreign key tables.</td>
</tr>
<tr>
<td>miv_functions</td>
<td>Extracts metadata for functions, including function signatures and bodies.</td>
</tr>
<tr>
<td>miv_parameters</td>
<td>Extracts metadata for function parameters.</td>
</tr>
</tbody>
</table>

The type attribute of these elements can be set to either SQLStatement (if using SQL to extract metadata) or API (if using Java metadata extraction).

In the content of the element, enter a SQL query to execute on the source. Use Oracle SQL if querying an Oracle database or using a gateway, or the native dialect of SQL if using JDBC connectivity.

The columns to return for each query and examples are defined in the Appendix, under “Query Elements.”

Each SQL query is passed the following four parameters:

- **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.

See the example queries below for how to reference the parameters using placeholders in your CMI queries.

**Building a SQL-Based CMI**

To build a SQL-based CMI definition:
• Create your CMI XML file (for example, name it myplatform_cmi.xml).
• Populate the `miv_capabilities` element with the types of metadata extraction that are available for this source, and the subsequent elements with the requisite queries.
• Use Oracle SQL for Oracle or gateway-based access, or platform-native SQL dialect for platforms accessed through JDBC.

**Basic SQL-Based CMI (No Business Areas)**

To create a basic SQL-based CMI:
• Define a CMI XML file that defines all the capability elements.
• Omit the `business_area` elements.
• Include all relevant SQL query elements defined in the reference.
• Omit the `miv_business_areas` and `miv_business_area_objects` queries.

**SQL-Based CMI with Business Areas**

For a CMI with business areas, you must add several additional elements to the basic SQL-based CMI definition.

In the `miv_capabilities` element:
• Add element `business_area_supported`, with content `true`.
• Add element `business_area_table_supported`, with content `true` or `false` if tables are grouped by business area.
• Add element `business_area_view_supported` with content `true` or `false` if views are to be grouped by business area.
• Add element `business_area_sequence_supported` with content `true` or `false` if sequences are to be grouped by business area.

To the query elements, add the `miv_business_areas` and `miv_business_area_objects` queries. See the CMI .element reference for columns and semantics.
Building a Java API-based CMI

The metadata returned by a Java API-based CMI is analogous to the metadata from a SQL-based CMI. However, it is limited to the extraction of tables, views, sequences, unique keys, and foreign key relationships. More complex metadata, such as function definitions and business areas are not supported.

You have to create a Java class that fulfills the interface

    oracle.wh.service.sdk.integrator.MetadataImport,

described below:

    public interface oracle.wh.service.sdk.integrator.MetadataImport{
        // The setup connection api is passed a number of properties from the
        // metadata location used for the import operation. It is entirely up to
        // the custom import code as to how these are to be used.
        // DATASERVER_NAME
        // USER
        // SCHEMA
        // PASSWORD
        // CATALOG
        // PORT
        // URL
        // DRIVER
        public abstract void setupConnection(java.util.Properties);
        public abstract void disconnect();
        public abstract java.util.List getUniqueKeys(java.lang.String,
            java.lang.String);
        public abstract java.util.List getForeignKeys(java.lang.String,
            java.lang.String);
        public abstract java.util.List getSources(oracle.wh.service.sdk.integrator.MetadataImport$SourceType,
            java.lang.String, java.lang.String);
        public abstract java.util.List getKeyColumns(java.lang.String,
            java.lang.String, java.lang.String);
        public abstract java.util.List getRelatedSources(java.lang.String,
            java.lang.String);

        // Return an entry in the list for each column. The entry should be an array of
        // strings containing, in order:
        // column name
        // type
        // length
        // precision
        // scale
        // description
        // nullable
        // type owner
        // default value
        // charset
        // bytes/char
        // character semantics
        public abstract java.util.List getColumns(java.lang.String, java.lang.String);
    }

Note the analogies between these API functions and the metadata returned by SQL CMI queries.

- The names returned from getSources() are exposed in OWB as tables, views or sequences based on the value of the initial
  oracle.wh.service.sdk.integrator.MetadataImport$SourceType parameter.
The inner class `SourceType` is defined as an enumeration:

```java
public final class oracle.wh.service.sdk.integrator.MetadataImport$SourceType
extends java.lang.Enum{
    public static final oracle.wh.service.sdk.integrator.MetadataImport$SourceType TABLE;
    public static final oracle.wh.service.sdk.integrator.MetadataImport$SourceType VIEW;
    public static final oracle.wh.service.sdk.integrator.MetadataImport$SourceType[] values();
    public static oracle.wh.service.sdk.integrator.MetadataImport$SourceType
.valueOf(java.lang.String);
}
```

- The strings returned by `getColumns()` are the same as the columns returned by the SQL-based CMI query. `getRelatedSources()` and `getKeyColumns()` map to `miv_fk_tables` and `miv_key_columns`, respectively.

- The connection properties passed to `setupConnection()` have the same names as the values used by OWB for a JDBC connection. You can give these values any semantics that are appropriate for your application—the names of the properties are in no way binding on how you use the actual values passed. `DATASERVER NAME`, `SCHEMA`, `CATALOG`, `PORT` and `URL` but could be used to pass any values useful for your particular source. You could also pass an arbitrary string in `URL` and parse it as needed for your data source, ignoring the other parameters.

Tip: For clarity and convenience, `USER` and `PASSWORD` are probably best used for any username and password credentials that apply in your application.

This flexibility is useful when supporting more exotic data sources. Examples could include:

<table>
<thead>
<tr>
<th>SOURCE TYPE</th>
<th>PARAMETER SEMANTICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAP server</td>
<td><strong>DATASERVER NAME</strong>: the email server</td>
</tr>
<tr>
<td>Extracting content from incoming emails</td>
<td><strong>PORT</strong>: Port to connect to</td>
</tr>
<tr>
<td></td>
<td><strong>CATALOG</strong>: mailbox name</td>
</tr>
<tr>
<td></td>
<td><strong>SCHEMA</strong>: Ignored</td>
</tr>
<tr>
<td>PDF document collection</td>
<td><strong>DATASERVER NAME</strong>: Path to a directory of PDF documents</td>
</tr>
<tr>
<td></td>
<td><strong>SCHEMA</strong>: A regular expression matching one or more filenames</td>
</tr>
<tr>
<td></td>
<td><strong>CATALOG</strong>: ‘METADATA’ (to extract metadata about the documents),</td>
</tr>
<tr>
<td></td>
<td>‘FORM’ (form responses to be extracted from the body of the document),</td>
</tr>
<tr>
<td></td>
<td>‘TEXT’ (main document content).</td>
</tr>
<tr>
<td>Twitter search API results</td>
<td><strong>USER</strong>: Name of the user account (if any) for the query</td>
</tr>
<tr>
<td>Run queries against the Twitter public APIs, and pull the results into a database table</td>
<td><strong>PASSWORD</strong>: password for the user account</td>
</tr>
<tr>
<td></td>
<td><strong>URL</strong>: A URL-encoded expression listing the remaining search parameters</td>
</tr>
<tr>
<td>RSS or Atom Feeds</td>
<td><strong>URL</strong>: URL of the feed to be requested</td>
</tr>
<tr>
<td>Parse items in the feed, pull the item contents into the database table</td>
<td><strong>SCHEMA</strong>: format of the feed to parse—‘RSS 0.91’, ‘Atom 1.0’, ‘RSS 2.0’</td>
</tr>
<tr>
<td></td>
<td><strong>USERNAME/PASSWORD</strong>: For authenticating the HTTP request.</td>
</tr>
</tbody>
</table>
Place the JAR file containing the classes you create for your CMI in the owb/lib/ext directory and restart Design Center so that they are loaded.

Creating the CMI Definition File for a Java CMI

A CMI definition file is required when using Java-based metadata extraction. You can get a sample Java CMI definition file at the OWB SDK page on OTN: http://www.oracle.com/technology/products/warehouse/SDK

As with a SQL CMI file, a `miv_capabilities` element describes the supported capabilities. Note that the business area-related capabilities must all be set to `false`. Java CMIs do not currently support business areas.

One new CMI element is added: `miv_java_class`, which specifies the name of the Java CMI class that implements the `oracle.wh.service.sdk.integrator.MetadataImport` interface.

Any CMI element (other than `miv_capabilities` and its children) that is present must have the `type` attribute set to `API`. The value of the default attribute is ignored. For example:

```xml
<mv>
  <miv_java_class type="API" default="true">
    oracle.wh.service.impl.integrator.jdbc.JdbcMetadataImport
  </miv_java_class>
  <miv_schemas type="API" default="true"/>
  <miv_tables type="API" default="true"/>
  <miv_views type="API" default="true"/>
  <miv_sequences type="API" default="true"/>
  <miv_columns type="API" default="true"/>
  <miv_unique_keys type="API" default="true"/>
  <miv_foreign_keys type="API" default="true"/>
  <miv_key_columns type="API" default="true"/>
  <miv_fk_tables type="API" default="true"/>
  <miv_capabilities type="ResultSet">
    <table_supported>true</table_supported>
    <view_supported>true</view_supported>
    <sequence_supported>false</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>false</table_fklevel_supported>
  </miv_capabilities>
</mv>
```
Associating a CMI Definition with an Application

To define a new application type with a CMI definition file, use the OMB*Plus OMBCREATE CMI_DEFINITION command:

```
OMBCREATE CMI_DEFINITION 'NEW_APP' USING DEFINITION_FILE 'new_app_miv.XML';
```

The new application is then visible in the Project Navigator in OWB Design Center.

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.

Associating a CMI Definition with a Database

To load the CMI definition into your repository and associate it with a platform definition, use OMBCREATE CMI_DEFINITION and set the MIV_TYPE property of the object to the value ‘databases’. For example:

```
OMBCREATE CMI_DEFINITION 'MSACCESS_IMPORT' USING DEFINITION_FILE 'miv_msaccess.xml'

OMBALTER CMI_DEFINITION 'MSACCESS_IMPORT' SET PROPERTIES (BUSINESS_NAME, MIV_TYPE) VALUES ('Microsoft Access', 'databases')

OMBALTER PLATFORM 'MSACCESS' SET REF CMI_DEFINITION 'MSACCESS_IMPORT'
```

Setting the MIV_TYPE property to ‘databases’ identifies this as a database CMI rather than an application CMI. As a result, the new platform appears in the Locations Navigator under Databases instead of Applications.

This example also supplies a business name for the CMI definition, which is used when displaying it in the tree with the other database locations.

Uninstalling and Updating a CMI Definition

If you need to modify your CMI definition, for example during development, simply delete all modules that you have created that reference the connector; then load the updated CMI definition.
Faking a Connection: The JDBC Stub Driver

For non-JDBC sources no JDBC driver is actually needed to extract metadata. However, it can be useful to provide a JDBC “stub driver” to enable the following features in OWB to work as expected:

- When working with locations associated with such sources, if there is no JDBC driver, clicking the “Test Connection” button for the location dialog box will result in an error. A stub driver that returns a valid response can prevent this error. This can help make your platform definition easier to use for developers who do not expect this error.
- If you are using an exotic source like those described above, and you provide your own JDBC driver, you could validate the parameters for the location when “Test Connection” is clicked.

Community Platform Definitions and CMI Definitions

Sample platform definitions and CMI definitions, as well as this paper by way of documentation, will be made available from the OWB SDK home page on OTN, here:


Also, some will be distributed using the “Check for Updates” mechanism built into the OWB Design Center.

Members of the community are encouraged to contribute useful platform definitions and CMI definitions. The surest way to bring information about these to the attention of OWB product management and development is to post a discussion on the OWB LinkedIn Group, accessible here:

http://www.linkedin.com/groups?home=&gid=140609

(If you are not already a member, join.) This will also bring your work to the attention of others in the OWB community.

Summary

This paper presents the tools you need to extend OWB to support a non-Oracle platform:

- How to define new platforms and platform types
- How to define new metadata extraction procedures
- How to define new application adapters
With the information in this whitepaper and the knowledge module technology as described in the Oracle Data Integrator documentation, customers can build their own solutions for loading Oracle data warehouses from nearly any data source, handling the data types as appropriate for the non-Oracle platform.

Oracle will make available new OWB platforms, created by Oracle and the OWB user community, on OTN, here:

http://www.oracle.com/technology/products/warehouse/SDK

You can make contact with the OWB user community through social media presences such as Oracle Mix or LinkedIn, here:

Appendix: MIV/CMI Element Reference

miv_capabilities Element

The following table describes the child elements of the miv_capabilities element of a CMI file.

<table>
<thead>
<tr>
<th>ELEMENT NAME</th>
<th>DESCRIPTION</th>
<th>DEFAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_supported</td>
<td>Whether this CMI supports the import of table metadata</td>
<td>true</td>
</tr>
<tr>
<td>view_supported</td>
<td>Whether this CMI supports import of view metadata</td>
<td>false</td>
</tr>
<tr>
<td>sequence_supported</td>
<td>Whether this CMI supports import of sequence metadata</td>
<td>false</td>
</tr>
<tr>
<td>table_name_filter_supported</td>
<td>Whether this CMI supports filtering table metadata import based on sequence names.</td>
<td>true</td>
</tr>
<tr>
<td>view_name_filter_supported</td>
<td>Whether this CMI supports filtering view metadata import based on sequence names.</td>
<td>false</td>
</tr>
<tr>
<td>sequence_name_filter_supported</td>
<td>Whether this CMI supports filtering sequence metadata import based on sequence names.</td>
<td>false</td>
</tr>
<tr>
<td>business_area_supported</td>
<td>Whether this CMI supports grouping imported objects by business area.</td>
<td>false</td>
</tr>
<tr>
<td>business_area_table_supported</td>
<td>Whether this CMI supports grouping imported tables by business area.</td>
<td>false</td>
</tr>
<tr>
<td>business_area_view_supported</td>
<td>Whether this CMI supports grouping imported views by business area.</td>
<td>false</td>
</tr>
<tr>
<td>business_area_sequence_supported</td>
<td>Whether this CMI supports grouping imported sequences by business area.</td>
<td>false</td>
</tr>
<tr>
<td>application_owner_supported</td>
<td>Whether this CMI exposes objects owned by multiple database-level users (e.g. if each application in a suite stores its objects under a different owner).</td>
<td>false</td>
</tr>
<tr>
<td>table_fklevel_supported</td>
<td>Whether this CMI supports the import of foreign keys.</td>
<td>true</td>
</tr>
<tr>
<td>reimport_supported</td>
<td>Whether reimport is supported.</td>
<td>true</td>
</tr>
<tr>
<td>data_object_at_leaf_levels</td>
<td>Whether data objects can be exposed at leaf levels in the tree, if business areas are in use.</td>
<td>true</td>
</tr>
<tr>
<td>multiple_tree_supported</td>
<td>Whether a single object can be exposed in multiple places in the business area tree (useful where the tree includes categories that overlap).</td>
<td>true</td>
</tr>
</tbody>
</table>

- For SQL queries, the element body contains the actual query. For example:

```xml
<miv_tables type="SQLStatement" default="true">
    SELECT rtrim(table_name) table_name, rtrim(table_name) business_name
    FROM all_tables atc
    WHERE owner = <Parameter name="owner"/>
</miv_tables>
```
• Queries can generally be omitted if the related capability is disabled. For example, if you do not intend to extract view metadata, and you set the view_supported capability to false, you do not need to define query MIV_VIEWS.

• Queries can return an empty set of rows.

Query Elements
The following query elements are created as children of the <miv> element in the MIV file.

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

This element determines how tables 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 parameter named 'owner' will be available within the query for the SQL template provided.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>Name of the table</td>
</tr>
<tr>
<td>business_name</td>
<td>Business name for the table</td>
</tr>
<tr>
<td>description</td>
<td>Description of the table</td>
</tr>
</tbody>
</table>

Example:

```xml
<miv_tables type="SQLStatement" default="true">
  SELECT rtrim(table_name) table_name, rtrim(table_name) business_name, 'no description' description
  FROM all_tables 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 table 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 parameter named 'owner' will be available within the query for the SQL template provided.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>view_name</td>
<td>Name of the view</td>
</tr>
<tr>
<td>business_name</td>
<td>Business name for the view</td>
</tr>
<tr>
<td>COLUMN NAME</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>description</td>
<td>Description of the view</td>
</tr>
</tbody>
</table>

Example:

```xml
<miv_views type="SQLStatement" default="true">
  SELECT rtrim(view_name) view_name,
          rtrim(view_name) business_name,
          'no description' description
  FROM all_tables atc
  WHERE owner = <Parameter name="owner"/>
</miv_views>
```

**Query MIV_COLUMNS**

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

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity_name</td>
<td>Name of the entity (e.g. table or view)</td>
</tr>
<tr>
<td>column_name</td>
<td>Name of the column</td>
</tr>
<tr>
<td>business_name</td>
<td>Business name of the column</td>
</tr>
<tr>
<td>description</td>
<td>Description of the column</td>
</tr>
<tr>
<td>position</td>
<td>Integer position of the column</td>
</tr>
<tr>
<td>datatype</td>
<td>Type of the column</td>
</tr>
<tr>
<td>length</td>
<td>Length of the column</td>
</tr>
<tr>
<td>precision</td>
<td>For numeric columns, the precision of the column</td>
</tr>
<tr>
<td>scale</td>
<td>For numeric columns, the scale of the column</td>
</tr>
<tr>
<td>nulls_allowed</td>
<td>Whether the column can have null values</td>
</tr>
<tr>
<td>charset</td>
<td>The IANA charset [RFC 2978] corresponding to the character set/code page defined for this attribute on the metadata source.</td>
</tr>
<tr>
<td>bytes_per_char</td>
<td>The (max) number of bytes required to represent a character in the charset. Integer, default = 1</td>
</tr>
<tr>
<td>use_char_semantics</td>
<td>Did the definition for this attribute on the metadata source specify character semantics? (Y/N). default N</td>
</tr>
</tbody>
</table>

Example:

```xml
<miv_columns type="SQLStatement" default="true">
  SELECT ac.table_name entity_name,
         rtrim(ac.column_name) column_name,
         rtrim(ac.column_name) business_name,
         '' description,
         ac.column_id position,
         rtrim(ac.data_type) data_type,
```
OWB 11.2 Platform and Application Adapter Extensibility Cookbook

```
ac.data_length length,
ac.data_precision precision,
ac.data_scale scale,
ac.nullable isNullable
FROM all_tab_columns ac
WHERE ac.owner = <Parameter name="owner"/>
ORDER BY ac.table_name, ac.column_id
</miv_columns>

Query MIV_SEQUENCES

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.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence_name</td>
<td>Name of the sequence</td>
</tr>
<tr>
<td>business_name</td>
<td>Business name for the sequence</td>
</tr>
<tr>
<td>description</td>
<td>Description of the sequence</td>
</tr>
</tbody>
</table>

Example:
```
<mi_v_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
</mi_v_sequences>
```

Query MIV_UNIQUE_KEYS

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

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity_name</td>
<td>Name of the entity</td>
</tr>
<tr>
<td>key_name</td>
<td>Key name</td>
</tr>
<tr>
<td>business_name</td>
<td>Business name</td>
</tr>
<tr>
<td>constraint_type</td>
<td>Type of the constraint (U=unique, P=primary)</td>
</tr>
<tr>
<td>description</td>
<td>Description of the table</td>
</tr>
</tbody>
</table>

Example:
### Query MIV_FOREIGN_KEYS

Extracts metadata for all foreign key relationships.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity_name</td>
<td>Name of the entity (typically, table)</td>
</tr>
<tr>
<td>foreign_key_name</td>
<td>Name of the foreign key</td>
</tr>
<tr>
<td>business_name</td>
<td>Business name for the table</td>
</tr>
<tr>
<td>description</td>
<td>Description</td>
</tr>
<tr>
<td>unique_key_name</td>
<td>Name of the unique key</td>
</tr>
</tbody>
</table>

**Example:**

```xml
<miv_foreign_keys type="SQLStatement" default="true">
  SELECT
    ac.table_name entity_name,
    rtrim(ac.constraint_name) foreign_key_name,
    rtrim(ac.constraint_name) business_name,
    '' description,
    rtrim(ac.r_constraint_name) unique_key_name
  FROM all_constraints 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.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>key_name</td>
<td>Name of the key</td>
</tr>
<tr>
<td>column_name</td>
<td>Name of the column storing the key</td>
</tr>
<tr>
<td>position</td>
<td>Integer position of the column in the table</td>
</tr>
</tbody>
</table>

**Example:**

```xml
<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
  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.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>orig_table_name</td>
<td>Name of the original table</td>
</tr>
</tbody>
</table>
foreign_table_name | Name of the Foreign table  
foreign_table_description | Description of the foreign table

Example:

```xml
<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 ac2,
        all_tab_comments atc,
        all_constraints 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>
```

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.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>business_area_id</td>
<td>ID of the business area.</td>
</tr>
<tr>
<td>parent_business_area_id</td>
<td>ID of the parent business area. Should be '0' for each top level business area.</td>
</tr>
<tr>
<td>business_area_name</td>
<td>Name for the business area</td>
</tr>
<tr>
<td>business_name</td>
<td>Name for the top level business area</td>
</tr>
<tr>
<td>description</td>
<td>Description</td>
</tr>
</tbody>
</table>

Example:

```xml
<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.

<table>
<thead>
<tr>
<th>COLUMN NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>

Example:

```xml
<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>
```
<table>
<thead>
<tr>
<th>object name</th>
<th>Name of the table, view or sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent_business_area_id</td>
<td>ID of the parent business area.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the object. Values can be: 'TABLE', 'VIEW', 'SEQUENCE'.</td>
</tr>
<tr>
<td>description</td>
<td>Description of the object.</td>
</tr>
</tbody>
</table>

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