



An Oracle White Paper
July 2011

What's The Diff? Using the Metadata Differ to Compare Object Metadata and Correct the Differences

Introduction	2
Problems and Requirements	3
SXML: a Friendlier XML for Object Metadata	4
Programming Details	8
The Metadata Diff Document.....	8
Prerequisites	9
Programming Details	10
Processing the Metadata Diff Document.....	10
Programming Details	12
Conclusion	14
Appendix: Metadata API Concepts	15
References.....	16

Introduction

Databases, like snowflakes, are never identical – especially when they're supposed to be. Two instances in the data center are behaving differently, but they're running the exact same software...aren't they? And if they aren't, then how are they different? What, so to speak, is the diff?

A full answer to this question requires, among other things, a comparison of the objects in the two databases: the tables, indexes, views and so on. In Oracle Database 11g release 2, Oracle introduces a new tool to aid in this task: the Metadata Differ. This tool consists of the following enhancements to the Metadata API¹ of Oracle Data Pump:

- New functionality in the existing DBMS_METADATA PL/SQL package
- A new PL/SQL package, DBMS_METADATA_DIFF

Together, these packages allow you to compare the metadata for two objects (for example, two tables or two indexes), and identify any differences. In addition, they can return a set of SQL ALTER statements that can be used to make one object like the other.

The features described in this paper have been used internally in a number of Oracle products, including the following:

- The Database Schema Object Deployment Utility (XDF) of Oracle Applications
- The dictionary synchronizations component of Oracle Enterprise Manager Grid Control Change Management Pack
- Oracle Data Pump

These interfaces are being published and made available to customers in Oracle Database 11g release 2.

¹¹ For background on the Metadata API see “Appendix: Metadata API Concepts”.

Problems and Requirements

The Metadata API of Oracle Data Pump first appeared in Oracle 9i. Before that time, users who wanted to gather complete metadata for an object faced the problem of having to cobble it together using separate queries against different catalog views. The job of converting all that metadata into creation DDL was another problem left to the user to figure out. Starting with Oracle 9i, users were able with a single function (DBMS_METADATA.GET_XML or GET_DDL), to fetch the complete metadata for a database object in either XML or creation DDL.

Reception of the Metadata API was positive, yet almost immediately customers began voicing other requirements. They wanted a comparison tool that would provide a simple way to do the following:

- Compare two objects
- Show how the objects differed
- Produce SQL ALTER statements to make one object like the other

Users also requested an XML representation of object metadata which could serve as a source file for database objects, much as a .c file serves as program source. In such a format, documents could be created, edited, compared, and put under source control. It was especially important to be able to edit existing documents and to create new documents from scratch.

Users had tried to use the XML produced by the Metadata API (that is, the XML returned by DBMS_METADATA.GET_XML – to be referred to as “full XML” for the remainder of this document), but had run into difficulties. Full XML is an XML representation of metadata stored in the tables of the Oracle data dictionary². As such it reflects the complexity (and in some cases, the opacity) of the underlying dictionary data:

- Structural complexity. The dictionary metadata is structured for ease of processing by the server, and at times this can result in considerable structural complexity. For example, the object that holds constraint definition metadata contains 32 sub-objects spread over multiple levels.
- Logical complexity. The dictionary metadata includes columns of bit-encoded binary data (for example, FLAGS, PROPERTY) which are copied as-is into full XML. The interpretation of the bit-patterns is not always straightforward.
- Instance-specific values. The dictionary tables are linked together with numeric values which serve as primary key/foreign key pairs. For example, every row in sys.obj\$ has a unique value of obj#, and this object number is used in other tables, such as sys.tab\$. Full XML dutifully includes all these object numbers, but they pose a problem to anyone wanting to edit the document or create a similar document from scratch. They also complicate the task of any tool attempting to compare object

² The syntax for full XML is not documented because it is an Oracle-internal format, not intended to be produced or parsed except by the Metadata API.

metadata, because two database objects, even if they are in different databases, will almost inevitably have different obj# values.

Analysis of both requirements – the need for a metadata comparison utility and the need for an editable XML representation for metadata – showed that they could be addressed with a single approach comprising three elements:

- A new, human-readable XML dialect: “SXML”. SXML documents can be created from scratch, edited, and easily compared.
- A difference engine to compare SXML documents and produce an output document showing the differences, a “metadata diff document.”
- A set of transforms to convert the difference document into SQL ALTER statements.

These elements are discussed in more detail in the following sections.

SXML: a Friendlier XML for Object Metadata

The simplest way to introduce SXML (the name is short for either “simplified XML” or “SQL XML”) is by example. Here is the creation DDL for the well-known table SCOTT.EMP as generated in Oracle Database 11g release 2.

```
SQL> select dbms_metadata.get_ddl('TABLE','EMP','SCOTT') from dual;
```

```
CREATE TABLE "SCOTT"."EMP"  
( "EMPNO" NUMBER(4,0) NOT NULL ENABLE,  
  "ENAME" VARCHAR2(10),  
  "JOB" VARCHAR2(9),  
  "MGR" NUMBER(4,0),  
  "HIREDATE" DATE,  
  "SAL" NUMBER(7,2),  
  "COMM" NUMBER(7,2),  
  "DEPTNO" NUMBER(2,0)  
) SEGMENT CREATION IMMEDIATE  
PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING  
STORAGE(INITIAL 16384 NEXT 16384 MINEXTENTS 1 MAXEXTENTS 505  
PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT  
CELL_FLASH_CACHE DEFAULT)  
TABLESPACE "SYSTEM"
```

Here is the start of the SXML for the same table:

```
SQL> select dbms_metadata.get_sxml('TABLE','EMP','SCOTT') from dual;
```

```
<TABLE xmlns="http://xmlns.oracle.com/ku" version="1.0">  
<SCHEMA>SCOTT</SCHEMA>  
<NAME>EMP</NAME>  
<RELATIONAL_TABLE>  
<COL_LIST>  
<COL_LIST_ITEM>
```

```

<NAME>EMPNO</NAME>
<DATATYPE>NUMBER</DATATYPE>
<PRECISION>4</PRECISION>
<SCALE>0</SCALE>
<NOT_NULL/>
</COL_LIST_ITEM>
<COL_LIST_ITEM>
  <NAME>ENAME</NAME>
  <DATATYPE>VARCHAR2</DATATYPE>
  <LENGTH>10</LENGTH>
</COL_LIST_ITEM>

```

etc.

You can see that SXML is intended to look as much as possible like a direct translation of SQL creation DDL into XML. The tag names (SCHEMA, NAME, DATATYPE, and so on) and structure correspond to names used in the *Oracle Database SQL Language Reference*, and the element contents are the same values you would specify in SQL. For a somewhat brutal contrast, here is the full XML for the EMPNO column:

```

<COL_LIST_ITEM>
  <OBJ_NUM>57425</OBJ_NUM>
  <COL_NUM>1</COL_NUM>
  <INTCOL_NUM>1</INTCOL_NUM>
  <SEGCOL_NUM>1</SEGCOL_NUM>
  <PROPERTY>0</PROPERTY>
  <NAME>EMPNO</NAME>
  <TYPE_NUM>2</TYPE_NUM>
  <LENGTH>22</LENGTH>
  <PRECISION_NUM>4</PRECISION_NUM>
  <SCALE>0</SCALE>
  <NOT_NULL>1</NOT_NULL>
  <CHARSETID>0</CHARSETID>
  <CHARSETFORM>0</CHARSETFORM>
  <BASE_INTCOL_NUM>1</BASE_INTCOL_NUM>
  <BASE_COL_TYPE>0</BASE_COL_TYPE>
  <CON>
    <OWNER_NUM>53</OWNER_NUM>
    <NAME>SYS_C004126</NAME>
    <CON_NUM>4126</CON_NUM>
    <OBJ_NUM>57425</OBJ_NUM>
    <NUMCOLS>1</NUMCOLS>
    <CONTYPE>7</CONTYPE>
    <ENABLED>1</ENABLED>
    <INTCOLS>1</INTCOLS>
    <MTIME>08-JUL-09</MTIME>
    <FLAGS>12</FLAGS>
  </CON>
  <SPARE1>0</SPARE1>
  <SPARE2>0</SPARE2>
  <SPARE3>0</SPARE3>
</COL_LIST_ITEM>

```

Oracle Database 11g release 2 provides APIs to get an SXML document for an object (the simplest is the browsing API, DBMS_METADATA.GET_SXML, shown above), but the SXML dialect is simple enough that SXML documents can be created or edited by hand.

Several principles have guided the design of SXML, including the following:

1. An SXML document is intended to look like a direct translation from SQL to XML, in order to be intuitive and human-readable. The element tags are drawn from SQL, and the document structure corresponds to the syntax documented in the *Oracle Database SQL Language Reference*.
2. An SXML document contains XML elements and element content. With the exception of the “xmlns” and “version” attributes on the top-level element, there are no XML attributes. This syntactic detail becomes important when looking at SXML difference documents.
3. SXML aims to provide only one syntax for any particular SQL feature. The goal is to make comparison easier and differences self-explanatory.

For an example of the last point, consider the syntax for constraint state. SQL has four keywords, ENABLE, DISABLE, VALIDATE and NOVALIDATE, with ENABLE VALIDATE being the default. Consequently, despite their syntactic differences, the primary key constraints in the following three table definitions are semantically identical:

```
create table t1(a number primary key);
create table t2(a number primary key enable);
create table t3(a number primary key enable validate);
```

SXML reduces this syntactic variety to two elements, <DISABLE/> and <NOVALIDATE/>. To specify ENABLE, you simply omit the <DISABLE/> tag, and to specify VALIDATE, you omit the <NOVALIDATE/> tag. Thus, in SXML the primary key definitions for all three tables are identical, as follows:

```
<PRIMARY_KEY_CONSTRAINT_LIST_ITEM>
  <COL_LIST>
    <COL_LIST_ITEM>
      <NAME>A</NAME>
    </COL_LIST_ITEM>
  </COL_LIST>
</PRIMARY_KEY_CONSTRAINT_LIST_ITEM>
```

Suppose the primary key is disabled, as shown in the following statement:

```
create table t4(a number primary key disable);
```

The SXML would then look as follows:

```
<PRIMARY_KEY_CONSTRAINT_LIST>
  <PRIMARY_KEY_CONSTRAINT_LIST_ITEM>
    <COL_LIST>
      <COL_LIST_ITEM>
        <NAME>A</NAME>
```

```
</COL_LIST_ITEM>  
</COL_LIST>  
<DISABLE/>  
<NOVALIDATE/>  
</PRIMARY_KEY_CONSTRAINT_LIST_ITEM>  
</PRIMARY_KEY_CONSTRAINT_LIST>
```

Oracle Database 11g release 2 provides SXML support for the following object types:

- AQ_QUEUE
- AQ_QUEUE_TABLE
- CLUSTER
- CONTEXT
- DB_LINK
- FGA_POLICY
- INDEX
- MATERIALIZED_VIEW
- MATERIALIZED_VIEW_LOG
- RLS_CONTEXT
- RLS_GROUP
- RLS_POLICY
- ROLE
- SEQUENCE
- SYNONYM
- TABLE
- TABLESPACE
- TRIGGER
- TYPE_SPEC
- TYPE_BODY
- USER
- VIEW

The amount of detail in the SXML varies depending on the object type and the level of detail in the Oracle dictionary. If PL/SQL code is present in the object definition (in a trigger, for example), the

code is returned in a single element, unparsed. In the following example, the trigger code is returned, unparsed, in the PLSQL_BLOCK element:

```
SQL> create trigger trig1 before insert on t1
2 begin
3 null;
4 end;
5 /
```

Trigger created.

```
SQL> select dbms_metadata.get_sxml('TRIGGER','TRIG1') from dual;
```

```
<TRIGGER xmlns="http://xmlns.oracle.com/ku" version="1.0">
<SCHEMA>SCOTT</SCHEMA>
<NAME>TRIG1</NAME>
<TRIGGER_TYPE>BEFORE</TRIGGER_TYPE>
<DML_EVENT>
<EVENT_LIST>
<EVENT_LIST_ITEM>
<EVENT>INSERT</EVENT>
</EVENT_LIST_ITEM>
</EVENT_LIST>
<SCHEMA>SCOTT</SCHEMA>
<NAME>T1</NAME>
</DML_EVENT>
<PLSQL_BLOCK>begin
null;
end;</PLSQL_BLOCK>
</TRIGGER>
```

Programming Details

The simplest way to fetch an SXML document is with the `DBMS_METADATA.GET_SXML` function. (See examples in the previous section.) `GET_SXML` is a browsing interface, implemented using the Metadata API programming interface (`OPEN`, `SET_FILTER`, `ADD_TRANSFORM`, and so on). Object metadata is fetched in full XML and then converted to SXML using the SXML transform. SXML itself can be converted to DDL using the `SXMLDDL` transform. For syntax details, see *Oracle Database PL/SQL Packages and Types Reference*. For programming examples, see the chapter “Using the Metadata APIs” in *Oracle Database Utilities*.

The Metadata Diff Document

As with the SXML dialect itself, the simplest way to introduce the SXML metadata diff document is with an example. First, create two tables, and then compare them:

```
SQL> create table tab1(a number, b varchar2(10));
```

```
SQL> create table tab2(b varchar2(20));
SQL> select dbms_metadata_diff.compare_xml('TABLE','TAB1','TAB2') from dual;
```

```
<TABLE xmlns="http://xmlns.oracle.com/ku" version="1.0">
  <SCHEMA>SCOTT</SCHEMA>
  <NAME value1="TAB1">TAB2</NAME>
  <RELATIONAL_TABLE>
    <COL_LIST>
      <COL_LIST_ITEM src="1">
        <NAME>A</NAME>
        <DATATYPE>NUMBER</DATATYPE>
      </COL_LIST_ITEM>
      <COL_LIST_ITEM>
        <NAME>B</NAME>
        <DATATYPE>VARCHAR2</DATATYPE>
        <LENGTH value1="10">20</LENGTH>
      </COL_LIST_ITEM>
    </COL_LIST>
  </RELATIONAL_TABLE>
</TABLE>
```

In this example there are two similar tables, TAB1 and TAB2. TAB1 has a number column “A” which is missing from TAB2. Both tables have a VARCHAR2 column “B” but the column lengths are different.

To compare the tables the browsing interface COMPARE_SXML from the DBMS_METADATA_DIFF package was used (new in Oracle Database 11g release 2). The diff document shows the union of the two source documents with XML attributes “src” and “value1” identifying the differences. The behavior of these attributes is as follows:

- If a node is in one document but not in the other, the node has the attribute "src". The attribute value is "1" or "2" depending on which source document contained the node. In the example above, TAB1 (listed first in the COMPARE_SXML call) is source document 1 and TAB2 is source document 2. Since column “A” is present in TAB1 and not in TAB2, the COL_LIST_ITEM element for that column is decorated with the XML “src” attribute: <COL_LIST_ITEM src="1">.
- For simple elements, if their contents do not match, then the element and its content is copied from the second source document, and the element is given the attribute "value1". The attribute value is the element content from the first source document. In the example above, the content of the LENGTH element for column “B” is 10 in document 1, 20 in document 2. This is shown as follows: <LENGTH value1="10">20</LENGTH>

Prerequisites

There are two prerequisites for use of this functionality:

- Oracle XML Database must be installed and enabled
- Use of the DBMS_METADATA_DIFF package requires a license for the Oracle Enterprise Manager Change Management option

Programming Details

One of the interesting uses for this technology is comparing tables in different databases. The `DBMS_METADATA_DIFF` package allows you to do that. For example, if `DBS1` and `DBS2` are database links, you can compare the metadata for `SCOTT.EMP` on the two databases as follows:

```
SQL> select dbms_metadata_diff.compare_sxml('TABLE','EMP','EMP','SCOTT','SCOTT','DBS1','DBS2') from dual;
```

(For API details, see *Oracle Database PL/SQL Packages and Types Reference*.)

Processing the Metadata Diff Document

Producing an SXML diff document is interesting, but what can you do with it?

The best way to answer that question is with an example. Using another browsing interface, `COMPARE_ALTER`, also from the `DBMS_METADATA_DIFF` package, compare the two tables from the previous example:

```
SQL> create table tab1(a number, b varchar2(10));
SQL> create table tab2(b varchar2(20));
SQL> select dbms_metadata_diff.compare_alter('TABLE','TAB1','TAB2') from dual;
ALTER TABLE "SCOTT"."TAB1" DROP ("A")
ALTER TABLE "SCOTT"."TAB1" MODIFY ("B" VARCHAR2(20))
ALTER TABLE "SCOTT"."TAB1" RENAME TO "TAB2"
```

The `COMPARE_SXML` interface used in the example in the previous section simply showed the differences between the two objects, whereas the `COMPARE_ALTER` interface shown here returns a set of SQL `ALTER` statements to make the first table like the second.

But would it be wise to blindly execute these SQL `ALTER` statements? Look at the three SQL statements returned in the example. The first statement drops a column from `TAB1`, including all data in the column. Are you sure you can do without that data? The third statement attempts to rename `TAB1` to `TAB2` – which is understandable: the difference in names is one of the differences between the tables. But if you tried to execute the statement, it would fail with the error “name is already used by an existing object.”

Clearly, the software can show differences between the metadata of the objects and can return SQL statements to correct these metadata differences, but the choice of which changes should be applied is a policy decision which can only be made by the user. Consequently, the Metadata API has no mechanism for automatically executing the SQL `ALTER` statements; that step is left up to you.

But suppose you knew what your policy was? (for example, don't drop columns, don't rename tables, all other changes are okay.) How could you write a program to compare two objects and modify them according to your policy?

The Metadata API has a feature to make that possible, the `ALTER_XML` document. This is an XML document that contains the SQL `ALTER` statements and, optionally, metadata about them. Your program makes calls to `DBMS_METADATA.SET_PARSE_ITEM` to specify what metadata to return. An example will make this more concrete. Call a function (the code for the function is given

below) that returns an ALTER_XML document which includes CLAUSE_TYPE and COLUMN_ATTRIBUTE metadata:

```
SQL> select compare_alter_xml('TABLE','TAB1','TAB2') from dual;
```

```
<ALTER_XML xmlns="http://xmlns.oracle.com/ku" version="1.0">
  <OBJECT_TYPE>TABLE</OBJECT_TYPE>
  <OBJECT1>
    <SCHEMA>SCOTT</SCHEMA>
    <NAME>TAB1</NAME>
  </OBJECT1>
  <OBJECT2>
    <SCHEMA>SCOTT</SCHEMA>
    <NAME>TAB2</NAME>
  </OBJECT2>
  <ALTER_LIST>
    <ALTER_LIST_ITEM>
      <PARSE_LIST>
        <PARSE_LIST_ITEM>
          <ITEM>CLAUSE_TYPE</ITEM>
          <VALUE>DROP_COLUMN</VALUE>
        </PARSE_LIST_ITEM>
      </PARSE_LIST>
      <SQL_LIST>
        <SQL_LIST_ITEM>
          <TEXT>ALTER TABLE "SCOTT"."TAB1" DROP ("A")</TEXT>
        </SQL_LIST_ITEM>
      </SQL_LIST>
    </ALTER_LIST_ITEM>
    <ALTER_LIST_ITEM>
      <PARSE_LIST>
        <PARSE_LIST_ITEM>
          <ITEM>CLAUSE_TYPE</ITEM>
          <VALUE>MODIFY_COLUMN</VALUE>
        </PARSE_LIST_ITEM>
        <PARSE_LIST_ITEM>
          <ITEM>COLUMN_ATTRIBUTE</ITEM>
          <VALUE> SIZE_INCREASE</VALUE>
        </PARSE_LIST_ITEM>
      </PARSE_LIST>
      <SQL_LIST>
        <SQL_LIST_ITEM>
          <TEXT>ALTER TABLE "SCOTT"."TAB1" MODIFY ("B" VARCHAR2(20))</TEXT>
        </SQL_LIST_ITEM>
      </SQL_LIST>
    </ALTER_LIST_ITEM>
    <ALTER_LIST_ITEM>
      <PARSE_LIST>
        <PARSE_LIST_ITEM>
          <ITEM>CLAUSE_TYPE</ITEM>
          <VALUE>RENAME_TABLE</VALUE>
```

```

    </PARSE_LIST_ITEM>
  </PARSE_LIST>
  <SQL_LIST>
    <SQL_LIST_ITEM>
      <TEXT>ALTER TABLE "SCOTT"."TAB1" RENAME TO "TAB2"</TEXT>
    </SQL_LIST_ITEM>
  </SQL_LIST>
</ALTER_LIST_ITEM>
</ALTER_LIST>
</ALTER_XML>

```

The document begins with OBJECT1 and OBJECT2 elements showing the objects to be compared, followed by an ALTER_LIST. Each ALTER_LIST item contains the following:

- a PARSE_LIST containing the metadata about the ALTER operation
- a SQL_LIST containing a list of SQL statements to perform the ALTER operation

Using standard programming interfaces, a program can traverse the document and, based on the PARSE_LIST information, decide whether or not to apply the ALTER operation.

Programming Details

Some readers may be interested in the code for the COMPARE_ALTER_XML program used in this section. The following is only a sample program, but it shows how to use the interfaces:

```

CREATE OR REPLACE FUNCTION compare_alter_xml(
    object_type  IN VARCHAR2,
    name1        IN VARCHAR2,
    name2        IN VARCHAR2,
    schema1      IN VARCHAR2 DEFAULT NULL,
    schema2      IN VARCHAR2 DEFAULT NULL,
    network_link1 IN VARCHAR2 DEFAULT NULL,
    network_link2 IN VARCHAR2 DEFAULT NULL)
RETURN CLOB IS
-- local variables
c1          CLOB;
c2          CLOB;
whandle     NUMBER;
thandle     NUMBER;
BEGIN
--
-- Fetch the Metadata Diff document
--
c1 := dbms_metadata_diff.compare_sxml(object_type,
    name1, name2, schema1, schema2,
    network_link1,
    network_link2);
--
-- Convert to an ALTER_XML document using the ALTERXML transform
-- Set the CLAUSE_TYPE and COLUMN_ATTRIBUTE parse items
--

```

```
whandle := dbms_metadata.openw(object_type);
thandle := dbms_metadata.add_transform(whandle, 'ALTERXML');
dbms_metadata.set_parse_item(whandle,'CLAUSE_TYPE');
dbms_metadata.set_parse_item(whandle,'COLUMN_ATTRIBUTE');

DBMS_LOB.CREATETEMPORARY(c2, TRUE );
dbms_metadata.convert(whandle, c1, c2);
dbms_metadata.close(whandle);
RETURN c2;
END;
/
```

Conclusion

In Oracle Database 11g release 2, Oracle introduces SXML and the Metadata Differ. For several years these features have been used internally within Oracle development where they have proved extremely useful. They provide an intuitive XML representation for object metadata and powerful tools for detecting metadata differences and resolving them. Now that they are available to customers, a broader range of users can voice requirements that will further enhance this functionality.

Appendix: Metadata API Concepts

The term “Metadata API” refers to those parts of Oracle Data Pump that deal with the fetching and manipulation of database object metadata. It is implemented by two PL/SQL packages: DBMS_METADATA (introduced in Oracle 9i) and DBMS_METADATA_DIFF (introduced in Oracle Database 11g release 2).

The Metadata API makes use of several Web standards including the following:

XML (Extensible Markup Language) is a universal format for structured documents and data on the Web. It uses tags to delimit pieces of data but leaves the interpretation of the tags up to the application. The Metadata API represents database object metadata in XML because in this format the metadata can be easily parsed and transformed.

XSL (Extensible Stylesheet Language) is a language for expressing stylesheets.

XSLT (XSL Transformations) is a language for transforming XML documents.

XPath (XML Path Language) is an expression language used by XSLT to access or refer to parts of an XML document.

The Metadata API uses XSLT stylesheets to transform XML documents containing database object metadata, sometimes to other XML documents, sometimes to SQL DDL.

XML Schema is a language for describing the structure, content and semantics of a set of XML documents. This includes what elements are (and are not) allowed at any point, their data types, number of occurrences, and so on. XML schemas are commonly used to validate that instance documents conform to the schema’s specifications. The Metadata API supplies XML Schemas which can be used to validate SXML documents.

For more on these standards see <http://www.w3.org>.

The Metadata API allows you to do the following:

fetch an object’s metadata as XML

transform the XML in a variety of ways (including converting it to DDL)

submit the XML to re-create the object

transform the XML for an object into a simpler XML format, called SXML; the SXML document can then be edited, or two SXML documents can be compared, optionally generating SQL ALTER statements for making one object like the other

For the purposes of this API, every entity in the database is modeled as an **object** which belongs to an **object type**. For example, the table SCOTT.EMP is an object; its object type is TABLE. When you fetch an object’s metadata you must specify the object type. When you compare two objects, they must be of the same object type.

In order to fetch a particular object or set of objects within an object type, you specify a **filter**. Different filters are defined for each object type. For example, two of the filters defined for the TABLE object type are SCHEMA and NAME; they allow you to say that you want the table whose schema is SCOTT and whose name is EMP.

The metadata is returned in an XML document. You can use the API to specify one or more **transforms** (XSLT stylesheets) to be applied to the XML. The API provides some predefined transforms including one named “SXML” which transforms the XML document into a simplified XML format. (A predefined transform name like “SXML” actually denotes a collection of stylesheets, each specific to an object type; at run time the Metadata API picks the appropriate stylesheet based on the object type.) Since the XSLT language permits conditional transformation based on input parameters, you can also specify **transform parameters** for the transforms you have added.

It is often desirable to access specific attributes of an object’s metadata, such as its name or schema. You could get this information by parsing the returned metadata, but the API provides another mechanism: you can specify **parse items**, specific attributes that will be parsed out of the metadata and returned in a separate data structure.

The Metadata API provides two styles of retrieval interface: one for programmatic use, the other for ad hoc browsing. (The browsing interfaces are implemented using the programmatic interfaces.) In this white paper most of the examples have used browsing interfaces, but the programming interfaces are quite simple and straightforward and provide considerable power and flexibility.

For more information on the Metadata API, see *Oracle Database PL/SQL Packages and Types Reference* and the “Using the Metadata APIs” chapter in *Oracle Database Utilities*.

References

1. *Oracle Database Utilities* (Part E10701)
2. *Oracle Database PL/SQL Packages and Types Reference* (Part E10577)



What's the Diff? Using the Metadata Differ

Author: Lee Barton

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



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2011, Oracle and/or its affiliates. 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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd. 1010

Hardware and Software, Engineered to Work Together