Oracle XML DB and Forms Integration

A Joint Bosch and Oracle White Paper
April 2006

Oracle XML DB and Forms Integration

Oracle XML DB and Forms Integration

INTRODUCTION

This paper is a case study on how Robert Bosch India Limited successfully developed and deployed an application using Oracle XML DB and Oracle Forms to process large amounts of complex measurement data collected from stringent testing of automotive parts. We will use code samples to walk through the steps of loading XML documents into Oracle XML DB, creating relational views over stored XML documents, and using Oracle Forms to display XML data through relational views.

Business Requirements

Robert Bosch India Ltd. is a wholly owned subsidiary of the Bosch Group, a well-known global automotive technology products manufacturer. When a customer orders a custom automotive part (e.g., a fuel injector), the contract would always include a detailed specification (e.g., temperature, engine speed, fuel volume). After a part sample is developed, test benches will be used to simulate an environment conforming to the specification for collecting and verifying measurements. Both the measurement instructions and the results are in XML format.

LaborAutomat is an application software, developed by Robert Bosch India, to store, process, and report measurement instructions and measurement results. With this application, a user can define measurement instructions to be transferred online to the test bench. The test bench will then collect measured results to be stored in a database. Finally, reports will be generated from the database for users to analyze the results.

Meeting Business Requirements

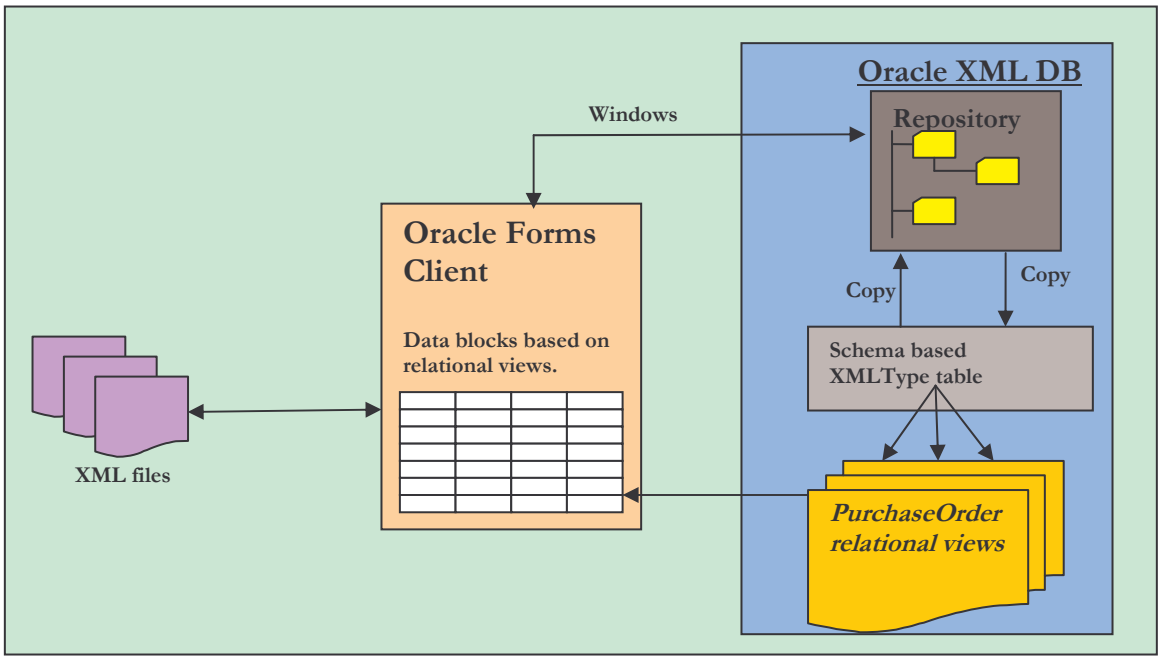
LaborAutomat has an online interface with the test bench for the purpose of bi-directional transferring of data between the testbench and the database. The data exchanged has a complex structure and therefore XML format was chosen for data representation and exchange. Two options were considered for processing the complex-structured XML data: Java or Oracle XML DB. Although custom Java code can always be developed to meet our business requirements, Oracle XML DB can handle the same tasks with much less coding and maintenance effort. Built on the solid foundation of Oracle database, Oracle XML DB provides extensive capabilities for the efficient storage, retrieval, generation, and management of large volumes of XML data. Based on the comparison below, we arrived at the decision that *LaborAutomat* will use Oracle XML DB for XML data processing.

XML DB	**Java**
XML data can be retrieved using standard SQL functions (Extract, ExtractValue, XMLTable, etc.)	Java code to read xml
XML data can be generated using standard SQL functions (XMLElement, XMLForest, XMLAgg, etc.)	Java code to generate XML
Database-native XML processing	Integrate Java Code with Oracle Forms as PJC(Pluggable Java Component)
Capability to create high performance relational views over natively stored XML data	Creation of temporary tables to store parsed data for display

The best part of Oracle XML DB is that all the storage and processing are database-native with superior performance and simplified application development, deployment, and maintenance. After quickly ramping up the initial learning curve, the entire business requirements were implemented using Oracle XML DB and Forms.

Application Architecture

The diagram below depicts the major components and processing flow of our application.



File Transfer between Clients and the Database

One of the main challenges to process files inside the database is to load them into the database. One option is to have a shared folder on the database server file system and then use DBMS_LOB or UTL_FILE to load files into the database. Since database administrators are usually hesitant to allow access to the database file system for security concerns, a better option would be to use *Oracle XML DB repository* as a temporary storage for transferring files between the client tier and the database tier.

Similar to a file system, *Oracle XML DB repository* allows file storage in the database as "files" in hierarchically-structured folders. These files and folders are referred to as *resources* and can be accessed using standard Internet protocols such as FTP, HTTP, and WEBDav. There are also data dictionary views (RESOURCE_VIEW and PATH_VIEW) that provide a mechanism for using SQL to access data stored in an Oracle XML DB repository. For complete details about Oracle XML DB repository, please refer to Oracle documentation: *Oracle XML DB Developers Guide*.

In the following sections, we walk through the essential processing steps of the *LaborAutomat* application.

STEPS TO DEVELOP THE LABORAUTOMAT APPLICATION

Step 1 – Creating a Repository Folder

As shown below, an XML DB repository folder resource can be created by a database user having the SYSDBA privilege.

```
DECLARE
 l_success         BOOLEAN := FALSE;
 l_resourceURL     RESOURCE_VIEW.ANY_PATH%TYPE := '/public/scott';
BEGIN
 IF (NOT DBMS_XDB.EXISTSRESOURCE(l_resourceURL)) THEN
     l_success := DBMS_XDB.CREATEFOLDER(l_resourceURL);
     COMMIT;
 END IF;
END;
```

Since folder creation is a DML operation, an explicit COMMIT is required to make the changes permanent.

Step 2 - Checking for the existence of a folder in the Oracle XML DB repository

To verify that a folder has indeed been created, we can use the following SQL statement to query the RESOURCE_VIEW dictionary view. Notice that resource names are case sensitive.

```
SELECT any_path
FROM resource_view
WHERE EQUALS_PATH (res, '/public/scott') =1
```

Step 3 - Registering an XML Schema

An XML Schema defines the structure, content, and other aspects of a set of XML documents. When an XML Schema is registered, it generates an object-relational storage structure (SQL types and tables) to natively store XML files inside the database. Oracle XML DB provides a flexible platform for XML storage mapping. For example,

- If the data is highly structured then each element in XML documents can be stored object-relationally as columns in database tables.
- If the data is unstructured then the data can be stored in CLOBs.

There are additional advantages of using XML schema to enable structured storage of XML documents:

- Validation of instance documents to ensure data quality
- XML documents are stored natively for scalability and high performance
- DOM fidelity to ensure preservation of information in XML documents
- Optimized collection management for ease of maintenance
- A number of indexing schemes are available for natively stored XML data to improve performance
- A object model can be automatically derived from an XML schema to simplify the mapping process

To reduce overhead, by default, Oracle XML DB does not conduct full schema validation when an XML document is stored into the database. However, there are XMLType methods for explicitly validating XML documents stored natively inside the database.

An XML Schema is registered using DBMS_XMLSCHEMA.RegisterSchema(). The SQL objects created during the schema registration process are case sensitive and you have to enclose them with double quotes while making direct references to them. To avoid this case sensitive problem you can always annotate the schema elements using attributes defined in Oracle XDB namespace: xmlns:xdb=http://xmlns.oracle.com/xdb.

To monitor the creation of object types, collections, and tables during XML schema registration, you can set the following event before invoking DBMS_XMLSCHEMA.RegisterSchema().

```
ALTER SESSION SET events='31098 trace name context forever';
```

Setting this event causes Oracle XML DB to log every CREATE TYPE and CREATE TABLE statements. The log file is created in the user session trace file, which can be found on the database server file system under the $ORACLE_HOME/admin/$ORACLE_SID/udump directory. You can also check the value of the USER_DUMP_DEST parameter to find this directory. This log file will be an useful aid in diagnosing problems with XML schema registration.

In the example below, we chose to manually create a table with an XMLType column and selected the object-relational storage option. This allows us to put in more information in the table to have more control over the XML documents based on other business logic.

```
--Register XML schema
DECLARE
  --assign the schema content to the variable here
  l_clob     CLOB := 'XML Schema Content';
BEGIN
 DBMS_XMLSCHEMA.REGISTERSCHEMA
    ('http://rbin.com/po.xsd'  --url to identify the schema in the database
    , l_clob                   --schema content
    , TRUE                     --local/global schema. in this case it is local
    , TRUE                     --generate types during registration
```

```
    , FALSE                    --do not generate beans
    , FALSE                    --do not generate default table for storage
    , FALSE                    --force option
    , USER);                   --owner
END;
/
--create table with a column based on XML schema
CREATE TABLE tab_xmldata
 (id          NUMBER PRIMARY KEY
 ,xml_data    SYS.XMLTYPE)
XMLTYPE COLUMN xml_data STORE AS OBJECT RELATIONAL
XMLSCHEMA "http://rbin.com/po.xsd"
ELEMENT "PurchaseOrder"
/
```

After an XML schema is successfully registered, whenever an XML file referencing a registered XML Schema is placed in the XML DB repository, Oracle automatically stores the XML instance into the default table created as a result of the XML schema registration (GEN_TABLES=>TRUE) process. You can then access the XML content from the default table. Similarly, whenever this file is deleted from the repository, the data is also deleted from the default table belonging to the XML schema.

Registered schema can be dropped by using DBMS_XMLSCHEMA.DeleteSchema(). During the delete operation, it has to first drop all the object tables and types created during the schema registration process. The force option is needed to ignore any inter-dependencies among object tables and SQL types. The example shows such an operation.

```
DBMS_XMLSCHEMA.DeleteSchema
( SCHEMA_URL      => 'http://rbin.com/po.xsd'
, DELETE_OPTION   =>  DBMS_XMLSCHEMA.DELETE_CASCADE_FORCE)
```

Below are two data dictionary views that provide information about the registered schemas and the created tables:

Useful views	Description
user_xml_schemas	lists all registered XML schema owned by the user
user_xml_tables	lists all the table belonging to a XML schema

Step 4 – Use FTP to move XML files

Now we are ready to FTP files to the XML DB repository and then copy them to the default table. In this section we will see how FTP scripts can be generated dynamically and how to perform FTP from Oracle Forms.

FTP script is generated on the client machine dynamically from Oracle Forms. Here is the code to do it. This procedure is to be added to the program unit of forms.

```
PROCEDURE FTP
( p_SourceFile    IN    VARCHAR2
, p_DestFile      IN    VARCHAR2
, p_mode          IN    VARCHAR2 DEFAULT 'PUT') IS

      ftp_host          VARCHAR2(200) := 'koredm106118';
      ftp_port          VARCHAR2(10) := '2100';
      ftp_user    VARCHAR2(32) := 'scott';
      ftp_pwd           VARCHAR2(32) := 'tiger';
      ftp_script_file   VARCHAR2(255);
      file_dir          VARCHAR2(255);
      file_handle TEXT_IO.FILE_TYPE;
  BEGIN
      --based on the mode, get the source/target file directory
      IF p_mode = 'PUT' THEN
            file_dir := substr(p_SourceFile, 1, instr(p_SourceFile, '\', -1));
      ELSIF p_mode = 'GET' THEN
```

```
            file_dir := substr(p_DestFile, 1, instr(p_DestFile, '\', -1));
      END IF;
      --generate the ftp script under the same directory
      --from where the file is to be uploaded/downloaded
      ftp_script_file := file_dir||'ftp_script.txt';

      --generate the ftp script file
      file_handle := TEXT_IO.FOPEN(ftp_script_file, 'W');
      TEXT_IO.PUT_LINE(file_handle, 'OPEN'||' '||ftp_host|| ' '||ftp_port);
      TEXT_IO.PUT_LINE(file_handle, 'USER'||' '||ftp_user|| ' '||ftp_pwd);
      TEXT_IO.PUT_LINE(file_handle, 'LCD' ||' '||rtrim(file_dir, '\'));
      TEXT_IO.PUT_LINE(file_handle, p_mode||' '||p_SourceFile||' '||p_DestFile);
      TEXT_IO.PUT_LINE(file_handle, 'CLOSE');
      TEXT_IO.PUT_LINE(file_handle, 'BYE');
      TEXT_IO.FCLOSE(file_handle);

      HOST('FTP -d -n -s:'||ftp_script_file);
  END FTP;
```

Here is a sample script file:

```
Open koredm106118 2100
User scott tiger
Lcd c:\myfiles
Put c:\myfiles\sample.xml /public/scott/sample.xml
Close
bye
```

FTP is conducted with a script file so that the entire process can be done without user intervention. The script will be generated under the same path from where the file is to be imported or exported. Once the script is available, the next step is to invoke it. It is done using HOST command available in Oracle Forms as shown in the example below.

```
HOST('ftp –d –n –s:c:\myfiles\ftp_script.txt');
```

Where `c:\myfiles\ftp_script.txt` is the FTP script file .

Options for FTP
-d - enables debuggig
-n - suppress autologin (because username and password is supplied in the script)
-s - indicates, ftp process will be done using FTP script file.

Please note that in this sample, we are using SCOTT and TIGER as the username and password for the FTP operations with the database. In actual you need not expose the username and password of the session user. A dummy database user with the `CREATE SESSION` privilege can be created exclusively for FTP operations.

To find out the port number for FTP operations, you can login as a SYSDBA-privileged user and execute the query below from SQL*PLUS.

```
SELECT ExtractValue(res,'//ftpconfig/ftp-port',
'xmlns="http://xmlns.oracle.com/xdb/xdbconfig.xsd"') ftp_port
 FROM resource_view
WHERE ANY_PATH LIKE '/xdbconfig.xml'
```

Step 5 - Copy Data from the Oracle XML DB Repository to an XML Schema-based table

After the FTP process is successful, the file is now inside the Oracle XML DB repository. To copy the contents to the table from repository, we used XDBURITYPE, which has methods like GetXML() and GetClob() that return XMLType content in desired format.

All database processing required for this paper is part of the PAK_UTILITY.PAK package that can be found in the separate download. Here is the code snippet from the package which does the task of copying the XML file from the repository to the table.

```
PROCEDURE copy_data_from_repos
(p_resourceURL IN resource_view.any_path%TYPE) IS
      l_xml         SYS.XMLTYPE;
      l_id          tab_xmldata.id%TYPE;

BEGIN
       SELECT  NVL(MAX(id), 0)+1  INTO l_id FROM tab_xmldata;

       /* extract the file contents from the repository */
       SELECT XDBURITYPE(p_resourceURL).GetXML() INTO l_xml FROM dual;

       INSERT INTO tab_xmldata VALUES (l_id, l_xml);

       /*delete the resource after copying the content */
       delete_resource_from_repos(p_resourceURL);

       COMMIT;
END copy_data_from_repos;
```

Step 6 - Creating relational views using XPATH and XQuery

Once you have the XML data in an XMLType column , the next step is to create views with XPath expressions. In this case we will create three views, `PURCHASEORDER_HEADER`, `SHIPPING_INSTRUCTIONS`, and `LINEITEMS`. The major advantage here is that once the views are created using XPATH, additional views can be created to hide any XML complexity.

To extract data from XMLType field, functions EXTRACT, EXTRACTVALUE, and XMLTABLE SQL functions are used as shown below.

Extract	Extract is used when the XPATH expression will result in a collection of nodes. The return type is XMLTYPE. The results of Extract can be either a document or a document fragment. For example,
	
	`SQL> SELECT EXTRACT(xml_data, '/PurchaseOrder/Reference') reference FROM tab_xmldata /`
	
	`REFERENCE ----------`
	`<Reference>AMCEWEN-20021009123336171PDT</Reference>`
	
	The above query returns an XML node. But if you want only the data without the element tags, you should use the text() function in the XPath.
	
	`SQL> SELECT EXTRACT(xml_data, '/PurchaseOrder/Reference/text()') reference FROM tab_xmldata /`
	
	`REFERENCE ----------`
	`AMCEWEN-20021009123336171PDT`

ExtractValue()	Returns the value of a single node or an attribute. For example,

```
SQL> SELECT EXTRACTVALUE(xml_data,
'/PurchaseOrder/Reference')  reference  FROM
tab_xmldata
/

REFERENCE
--------------------------------------------------
AMCEWEN-20021009123336171PDT
```

XMLTable() SQL Function

XMLTABLE() is an SQL/XML standard function to allow processing of XQuery expressions through SQL. XMLTABLE takes advantage of the power and flexibility of both XQuery and SQL. Using This function you can construct XML data using relational data , query relational data as if it were XML, and construct relational data from XML.

You can use XMLTABLE() to transform the result of an XQuery evaluation into relational rows and columns of a virtual table. You can then insert into a pre-existing database table, or you can query it using SQL-in a join expression. This function can be used in the FROM clause of the SQL statement. Prior to Oracle Database 10g Release 2, the functionality of XMLTABLE() had to be achieved using a more cumbersome combination of TABLE(), XMLSEQUENCE(), and EXTRACT() functions.

The example below is the more cumbersome approach using Oracle Database 9i Release 2 and Oracle Database 10g Release 1

```
SELECT extractValue(value(t),'/Description')
FROM tab_xmldata X,
TABLE ( xmlsequence
(extract(xml_data,'/PurchaseOrder/LineItems/LineItem/Description'))) t;
```

Here is an example of how XMLTable() can be used in Oracle Database 10g Release 2.

```
SELECT description
FROM tab_xmldata ,
    xmltable('/PurchaseOrder/LineItems/LineItem/Description' passing xml_data
             columns
             description varchar2(64) path '/')
```

Code sample below demonstrates how we can create XMLType views.

```
SET ECHO ON

CREATE OR REPLACE VIEW  purchaseorder_header_xml AS
SELECT  id
   ,EXTRACTVALUE(xml_data, '/PurchaseOrder/Reference')           reference
   ,EXTRACTVALUE(xml_data, '/PurchaseOrder/Requestor')           requestor
   ,EXTRACTVALUE(xml_data, '/PurchaseOrder/User')                po_user
   ,EXTRACTVALUE(xml_data, '/PurchaseOrder/CostCenter')          costcenter
   ,EXTRACTVALUE(xml_data, '/PurchaseOrder/SpecialInstructions') spl_instructions
FROM tab_xmldata
/
CREATE OR REPLACE VIEW purchaseorder_header
AS SELECT * FROM purchaseorder_header_xml
/

CREATE OR REPLACE VIEW shipping_instructions_xml AS
SELECT  id, name, address, telephone
FROM tab_xmldata
   , XMLTABLE('/PurchaseOrder/ShippingInstructions' passing xml_data
              COLUMNS
              name       VARCHAR2(64)   path   'name'
             ,address    VARCHAR2(64)   path   'address'
             ,telephone  VARCHAR2(10)   path   'telephone')
```

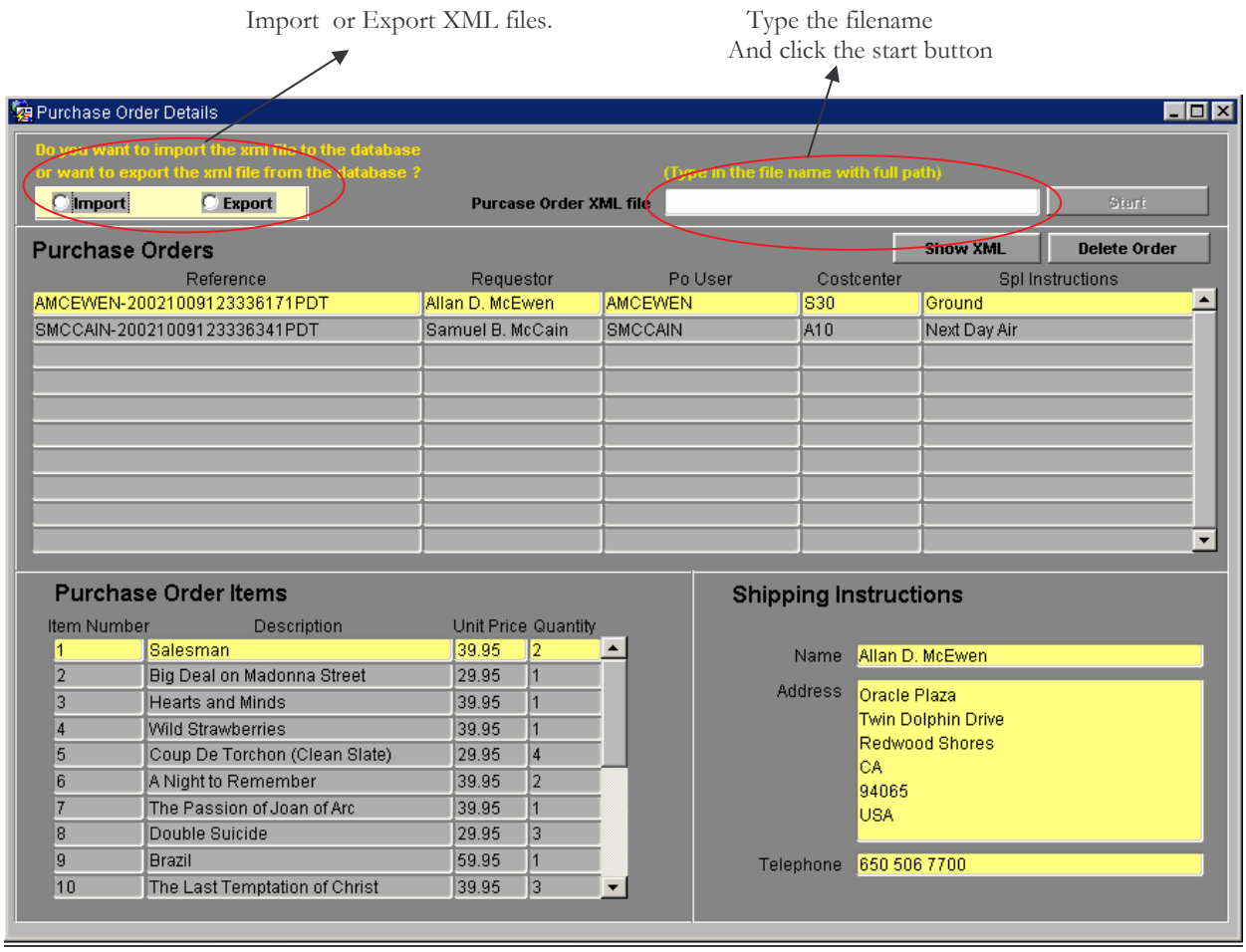
```
/

CREATE OR REPLACE VIEW shipping_instructions
AS SELECT * FROM shipping_instructions_xml
/

CREATE OR REPLACE VIEW line_items_xml AS
SELECT  id, partid, item_number, description, unit_price, quantity
FROM tab_xmldata
   , XMLTABLE('/PurchaseOrder/LineItems/LineItem' passing xml_data
             COLUMNS
              partid       VARCHAR2(15)  path   'Part/@Id'
             ,item_number  VARCHAR2(30)  path   '@ItemNumber'
             ,description  VARCHAR2(45)  path   'Description'
             ,unit_price   VARCHAR2(15)  path   'Part/@UnitPrice'
             ,quantity     NUMBER(5,2)   path   'Part/@Quantity')
/

CREATE OR REPLACE VIEW line_items
AS SELECT * FROM line_items_xml
/
```

In the above DDL, there are XXX_XML views, which contains the XPATH expressions, and there are other relational views created on top of these XXX_XML views. The reason is that Oracle Forms at the moment does not allow data blocks to be based on views containing XPath. As a workaround, additional relational views are created.

The next step is to develop the GUI to display the data. This GUI will be developed using Oracle Forms as the main purpose of this paper is to show Oracle Forms developers how to integrate Oracle XML DB based on the views created in **step 6.** However, we will not discuss the steps to create the forms in this paper.

Step 7 - Form to display data

Here is the sample form that can be downloaded with other code.



INSTALLING AND RUNNING THE DEMO

Please note that his paper does not deal with Oracle XML DB installation of configuration, it is assumed that you are working with oracle database 9i rel 2 and above and XML DB is installed and configured on your database. Your database listener is configured to run on default port of 1521 and it also accepts FTP requests.

- Configure or install XML DB in the database
- Execute the `create_resource.sql` with `SYS` account or with user having SYSDBA privilege.
- Register the XML schema in the database using `register_schema.sql`. This can be done under scott user or any other demo user.
- Create the table `tab_xmldata` using `create_objects.ddl`
- Create XML views using `xml_views.sql`.
- Create the utility package `pak_utility.pak`

The above scripts can be run against Oracle database 9.2.0.5 and above.

Steps to run the Oracle Forms demo

The Import Process

- Choose the **Import** radio button on the top left of the screen.
- Type in the XML file name with the full path, in the **purchase order XML file** field.
- Click the **start import** button.

The Export Process

- Choose the **Export** radio button on the top left of the screen.
- Type in the XML file name with the full path, in the **purchase order XML file** field.
- Select the purchase order record from the **Purchase Orders** block for which the XML file has to be exported.
- Click the **start Export** button.

View XML content

- Select the purchase order record from the **Purchase Orders** block for which the XML file has to be viewed.
- Click the **Show XML** button**.**

Delete a Purchase Order from the database

- Select the purchase order record from the **Purchase Orders** block for which the XML file has to be exported.
- Click the **Delete Order** button**.**

CONCLUSION

Using XML DB features, we were able to represent complex-structured XML data in a relational format and integrate with an Oracle Forms application. This integration provides a simple, flexible, and powerful solution to process XML files without writing a single line of Java code or accessing the file system on the database server. Most importantly, the entire processing flow takes place natively inside the database and the results are effectively displayed using Oracle Forms. This provides huge performance gains when large volumes of XML files have to be processed.

In short, Oracle XML DB has not only met our business requirements in high performance XML processing, it has also provided us with a rich set of database-native capabilities to simplify and accelerate the development, deployment, and maintenance effort of a critical business application.

Oracle XML DB and Forms Integration
April 2006
Author: Raghu Shyam T, Technical Manager, Robert Bosch India Limited
Contributing Authors: Geoff Lee, Principal Product Manager, Oracle Corporation

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
www.oracle.com