Oracle Multimedia Image Java Quick Start

Introduction

Oracle Multimedia is a feature that enables Oracle Database to store, manage, and retrieve images, audio, video, and other heterogeneous media data in an integrated fashion with other enterprise information. Oracle Multimedia extends Oracle Database reliability, availability, and data management to multimedia content in media-rich applications.

This article describes quickstart_image.java which provides simple examples that upload, store, manipulate, and export image data inside a database using Java and a table with a BLOB column. The article is divided into several sections, each explaining a specific method or aspects of the complete java example provided. Some common pitfalls are also highlighted. The PL/SQL package used here is available in Oracle Database release 12*c* Release 2 or later with Oracle Multimedia installed (the default configuration provided by Oracle Universal Installer). The functionality in this PL/SQL package is the same as the functionality available with the Oracle Multimedia relational interface. For more details refer to *Oracle Multimedia Reference* and *Oracle Multimedia User's Guide.*

NOTE: The following examples also connect to the database using

```
connect scott
Enter password: password
```

which you should change to an actual user name and password on your system.

Creating a Table with an Image BLOB column

First, we create a simple table with two columns: a numeric identifier (`id`), and a Binary Large OBject "BLOB" to hold the image itself (`image_blob`).

```
connect scott
Enter password: password

create table image_blob_table ( id number primary key, image_blob BLOB)
       lob(image_blob) store as securefile;
```

The included script, `quickstart_java_setup.sql` creates the above table.

Setting up the required Java environment

To connect to the database and use JDCB objects, the following jar file must be in your CLASSPATH:

a. $ORACLE_HOME/jdbc/lib/ojdbc7.jar (for Java 1.7)

b. $ORACLE_HOME/jdbc/lib/ojdbc6.jar (for Java 1.6)

To use the required JDBC classes in your Java program, the following import statements must be present:

```
import java.io.Console;
import java.io.FileNotFoundException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.SQLException;
```

```
import java.util.HashMap;
import java.util.Map;

import oracle.jdbc.OracleCallableStatement;
import oracle.jdbc.pool.OracleDataSource;
import oracle.jdbc.OracleConnection;
import oracle.jdbc.OracleTypes;
import oracle.sql.BLOB;
```

You need to create an instance of the `quickstart_image` class in order to access the methods described in this guide.

```
//instantiates a quickstart_image object
quickstart_image quickstart = new quickstart_image();
```

Creating the JDBC Connection

Before you can issue SQL statements in your Java programs, you must open a database connection. The `getConnection()` method initializes a JDBC Connection object, which is stored in your program so it can be referenced later. The Connection established must have the `autoCommit` flag set to false because Oracle Multimedia uses BLOB columns to store data. Since BLOB updates in Oracle Database require a two-stage select-update process, if the `autoCommit` flag is set to `true` (the default) then BLOB operations will fail.

```
public void getConnection(String connectURI )
{
       final OracleDataSource ods;
       String username;
       String password;

       try {
              Console cons= System.console();
              username = new String(cons.readLine("%s", "Username:"));
              password = new String(cons.readPassword("%s", "Password:"));

              ods = new OracleDataSource();
              ods.setURL(connectURI);
              ods.setUser(username);
              ods.setPassword(password);

              connection = (OracleConnection)ods.getConnection();

              //Set the autocommit to false.
              connection.setAutoCommit(false);
       } catch (SQLException e) {
              System.err.println("Unable to connect to the database.");
              System.err.println(e.getLocalizedMessage());
              System.exit(1);
       }
}
```

Call the `getConnection()` method passing the connection String as parameter.

```
//Create the connection.
quickstart.getConnection("jdbc:oracle:oci8:@");
```

After calling the `getConnection` method, your java program will prompt the user for a username and password as follows:

```
Username: username
Password: password
```

At this point you should specify the username and password for the user under which you ran the

`quickstart_java_setup.sql` script.

Importing images into the Database

This section shows how to import images from the file system into the newly created `image_blob_table`. Create the method `writeImageToDatabase()` that inserts a new row into the `image_blob_table` and then imports the image data into the newly created BLOB locator.

```
public void writeImageToDatabase(int id, String fileName )
              throws SQLException, IOException
              {
       //Define the PL/SQL block to insert the new row.
       final String INSERT_BLOB = "DECLARE "
                     + "    src_id       NUMBER; "
                     + "BEGIN "
                     + "    src_id := ?;"
                     + "    DELETE FROM image_blob_table WHERE id=src_id; "
                     + "    INSERT INTO image_blob_table t (id, image_blob) "
                     + "           VALUES(src_id, empty_blob()) "
                     + "           RETURNING t.image_blob INTO ?; "
                     + "END;";
       try {
              //Create the statement object.
              final OracleCallableStatement pstmt =
                     (OracleCallableStatement)connection.prepareCall(INSERT_BLOB);

              //Binding the variables to the statement.
              pstmt.setInt(1, id); //ID
              pstmt.registerOutParameter(2, OracleTypes.BLOB);
              pstmt.execute(); //Execute the PL/SQL statement.

              //Get the BLOB locator from the table.
              BLOB blob = pstmt.getBLOB(2);
              File binaryFile = new File(fileName);
              FileInputStream instream = new FileInputStream(binaryFile);

              //Retrieve the ideal buffer size to use in writing to the BLOB.
              int size = 1024*1024; // 1MB.
              byte[] buffer = new byte[size];
              int read = -1;
              long position =1;

              //Read the file to the byte array buffer, then write it to the BLOB.
              while ((read = instream.read(buffer)) != -1)
              {
                     blob.setBytes(position,buffer,0,read);
                     position+=read;
              }
              instream.close();
              connection.commit();

       } catch (FileNotFoundException e) {
              throw new FileNotFoundException("File " + fileName +" not Found.");
       } catch (IOException e) {
              throw new IOException("Error while reading " + fileName);
       }
              }
```

Call the `writeImageToDatabase()` method passing the row id and source file path.

```
//Write data from a local file into a BLOB in the database.
quickstart.writeImageToDatabase(1, "flowers.jpg");
```

NOTE: If the `autoCommit` flag on the connection is set to `true`, or is not set (the default is `true`), the following

error is returned when you attempt to select a row with a BLOB column for update:

```
java.sql.SQLException: ORA-22990: LOB locators cannot span transactions
```

Retrieving Image Properties

Once the image data has been imported from the file system into `image_blob_table`, the database does not know what the binary bytes in the `image_blob` BLOB column represent. In the following example, we show how to use the `ORDSYS.ORD_IMAGE.getProperties()` procedure of the Oracle Multimedia PL/SQL package to extract the image's properties into the Java Application.

Create the method `getProperties_example_j()` to extract the image's properties and return a HashMap object containing the information.

```
public HashMap<String, Object> getProperties_example_j(int id )
              throws SQLException
              {
       //Define the PL/SQL block to extract the properties.
       final String getPropertiesStmt = "DECLARE "
                     + "    src                         BLOB; "
                     + "    img_mimeType                VARCHAR2(32); "
                     + "    img_width                   INTEGER; "
                     + "    img_height                  INTEGER; "
                     + "    img_contentLength           INTEGER; "
                     + "    img_fileFormat              VARCHAR2(32); "
                     + "    img_contentFormat           VARCHAR2(32); "
                     + "    img_compressionFormat       VARCHAR2(32); "
                     + "BEGIN "
                     + "    SELECT image_blob INTO src FROM image_blob_table"
                     + "           WHERE id=?; "
                     + "    ORDSYS.ORD_IMAGE.getProperties(src, "
                     + "                   img_mimeType, "
                     + "                   img_width, "
                     + "                   img_height, "
                     + "                   img_fileFormat, "
                     + "                   img_compressionFormat, "
                     + "                   img_contentFormat, "
                     + "                   img_contentLength); "
                     + "    ? := img_mimeType; "
                     + "    ? := img_width; "
                     + "    ? := img_height; "
                     + "    ? := img_contentLength; "
                     + "    ? := img_fileFormat; "
                     + "    ? := img_contentFormat;"
                     + "    ? := img_compressionFormat; "
                     + "END;";

       //Create the statement object.
       final OracleCallableStatement pstmt =
              (OracleCallableStatement)connection.prepareCall(getPropertiesStmt);

       //Binding the variables to the statement.
       pstmt.setInt(1, id);
       pstmt.registerOutParameter(2, OracleTypes.VARCHAR);
       pstmt.registerOutParameter(3, OracleTypes.INTEGER);
       pstmt.registerOutParameter(4, OracleTypes.INTEGER);
       pstmt.registerOutParameter(5, OracleTypes.INTEGER);
       pstmt.registerOutParameter(6, OracleTypes.VARCHAR);
       pstmt.registerOutParameter(7, OracleTypes.VARCHAR);
       pstmt.registerOutParameter(8, OracleTypes.VARCHAR);

       //Execute the statement.
       pstmt.execute();

       //Create a HashMap object and populate it with the properties.
       HashMap<String, Object> map = new HashMap<String, Object>();
```

```
        map.put("mimeType", pstmt.getString(2));
        map.put("width", pstmt.getInt(3) );
        map.put("height", pstmt.getInt(4));
        map.put("contentLength", pstmt.getInt(5));
        map.put("fileFormat", pstmt.getString(6));
        map.put("contentFormat", pstmt.getString(7));
        map.put("compressionFormat", pstmt.getString(8));

        return map;
               }
```

Call the `getProperties_example_j()` method passing the id, then iterate over the HashMap to print the properties.

```
System.out.println("Original image properties");
HashMap<String, Object> attributesMap= quickstart.getProperties_example_j(1);
//Iterate over the HashMap.
for (Map.Entry<String, Object> entry : attributesMap.entrySet()) {
       System.out.println(entry.getKey() + " = " + entry.getValue());
}
```

This section of the program generates the following output for the `flowers.jpg` sample image.

```
Original image properties
compressionFormat = 24BITRGB
height = 1704
contentFormat = JPEG
width = 2272
contentLength = 1693670
mimeType = image/jpeg
fileFormat = JFIFo
```

NOTE: If the image data that is in the `image_blob` column is not one of Oracle Multimedia's supported formats (for example PSD) the following error is returned.

```
Exception in thread "main" java.sql.SQLException: ORA-29400: data cartridge error
```

Creating Thumbnails

We next illustrate some image processing operations that can be invoked within the database. To generate a thumbnail image from an existing image, the developer may use the `ORDSYS.ORD_IMAGE.thumbnail()` procedure of the Oracle Multimedia PL/SQL package. The following code allows us to obtain a thumbnail image from a source BLOB.

Create the method `thumbnail_example_j()` to generate a thumbnail from the image in the `src_id` row, and store it into the `dst_id` row.

```
public void thumbnail_example_j(int src_id, int dst_id ) throws SQLException
{
       //Define the PL/SQL block to create a thumbnail.
       final String createThumbnailStmt  = "DECLARE "
                    + "    src_blob     BLOB;"
                    + "    dst_blob     BLOB;"
                    + "    src_id       NUMBER;"
                    + "    dst_id       NUMBER;"
                    + "BEGIN"
                    + "    src_id := ?;"
                    + "    dst_id := ?;"
                    + "    DELETE FROM image_blob_table WHERE id = dst_id;"
                    + "    INSERT INTO image_blob_table(id, image_blob) "
                    + "           VALUES (dst_id, empty_blob()) "
```

```
                    + "          RETURNING image_blob INTO dst_blob;"
                    + "    SELECT image_blob INTO src_blob FROM image_blob_table"
                    + "          WHERE id = src_id;"
                    + "    ORDSYS.ORD_IMAGE.thumbnail(src_blob,dst_blob);"
                    + "    UPDATE image_blob_table SET image_blob = dst_blob"
                    + "          WHERE id = dst_id; "
                    + "END;";
        final OracleCallableStatement pstmt =
              (OracleCallableStatement)connection.prepareCall(createThumbnailStmt);

        //Binding the variables to the statement.
        pstmt.setInt(1, src_id);
        pstmt.setInt(2, dst_id);
        //Execute the statement.
        pstmt.execute();
        connection.commit();
}
```

Call the `thumbnail_example_j()` method passing the source and destination id.

```
//Create a thumbnail.
quickstart.thumbnail_example_j(1,2);
```

Changing Format

Next we use the `ORDSYS.ORD_IMAGE.convert()` procedure of the Oracle Multimedia PL/SQL package in order to create a new image with a different file format than the source image.

NOTE: Some image file extensions and the corresponding Oracle Multimedia `fileformat` values are as follows.

Extension	`fileformat`
.jpg	JFIF
.gif	GIFF
.tif, .tiff	TIFF
.png	PNGF

Create the method `convert_example_j()` to convert an image to a different file format and write a new row in `image_blob_table` with the new image.

```
public void convert_example_j(int src_id, int dst_id, String fileFormat )
              throws SQLException
              {
        //Define the PL/SQL block to change the file format.
        final String convertStmt = "DECLARE "
                    + "    src_blob     BLOB;"
                    + "    dst_blob     BLOB;"
                    + "    src_id       NUMBER;"
                    + "    dst_id       NUMBER;"
                    + "BEGIN"
                    + "    src_id := ?;"
                    + "    dst_id := ?;"
                    + "    DELETE FROM image_blob_table WHERE id=dst_id;"
                    + "    INSERT INTO image_blob_table(id, image_blob) "
                    + "          VALUES (dst_id, empty_blob()) "
                    + "          RETURNING image_blob INTO dst_blob;"
                    + "    SELECT image_blob INTO src_blob FROM image_blob_table"
                    + "          WHERE id=src_id; "
                    + "    ORDSYS.ORD_IMAGE.convert(src_blob,?,dst_blob); "
                    + "    UPDATE image_blob_table SET image_blob = dst_blob "
                    + "          WHERE id = dst_id;"
                    + "END;";

        final OracleCallableStatement pstmt =
```

```
                (OracleCallableStatement)connection.prepareCall(convertStmt);

        //Binding the variables to the statement.
        pstmt.setInt(1, src_id);
        pstmt.setInt(2, dst_id);
        pstmt.setString(3, fileFormat);
        pstmt.execute();
               }
```

Call the `convert_example_j()` method passing the connection object, the source id, the destination id, and the new file format, then call again the `getProperties_example_j()` method to verify the change.

```
//Change the file format to PNG.
quickstart.convert_example_j(1,3, "PNGF");

System.out.println("\nConverted image properties:");
attributesMap= quickstart.getProperties_example_j(3);
for (Map.Entry<String, Object> entry : attributesMap.entrySet()) {
       System.out.println(entry.getKey() + " = " + entry.getValue());
}
```

This section of the program generates the following output.

```
Converted image properties
compressionFormat = 24BITRGB
height = 600
contentFormat = DEFLATE
width = 800
contentLength = 702636
mimeType = image/png
fileFormat = PNGF
```

Crop an Image

We use the `ORDSYS.ORD_IMAGE.crop()` procedure of the Oracle Multimedia PL/SQL package in order to create a new image by defining a window to crop from the original image and writing the result into a new row.

Create the method `crop_example_j()` to crop the image in the `src_id` row, and store it into the `dst_id` row, the cropped section will be defined by the given coordinates and size: `originX`, `originY`, `width` and `height`.

```
public void crop_example_j(int src_id, int dst_id,
              int originX, int originY, int width, int height) throws SQLException
              {
        final String cropStmt  = "DECLARE "
                      + "    src_blob     BLOB;"
                      + "    dst_blob     BLOB;"
                      + "    src_id       NUMBER;"
                      + "    dst_id       NUMBER;"
                      + "BEGIN"
                      + "    src_id := ?;"
                      + "    dst_id := ?;"
                      + "    DELETE FROM image_blob_table WHERE id = dst_id;"
                      + "    INSERT INTO image_blob_table(id, image_blob) "
                      + "           VALUES (dst_id, empty_blob()) "
                      + "           RETURNING image_blob INTO dst_blob;"
                      + "    SELECT image_blob INTO src_blob FROM image_blob_table"
                      + "           WHERE id = src_id;"
                      + "    ORDSYS.ORD_IMAGE.crop(src_blob,?,?,?,?,dst_blob);"
                      + "    UPDATE image_blob_table SET image_blob = dst_blob"
                      + "           WHERE id = dst_id; "
                      + "END;";

        final OracleCallableStatement pstmt =
                      (OracleCallableStatement)connection.prepareCall(cropStmt);
```

```
        //Binding the variables to the statement.
        pstmt.setInt(1, src_id);
        pstmt.setInt(2, dst_id);
        pstmt.setInt(3, originX);
        pstmt.setInt(4, originY);
        pstmt.setInt(5, width);
        pstmt.setInt(6, height);

        //Execute the statement.
        pstmt.execute();
        connection.commit();
              }
```

Call the `crop_example_j()` method passing the source and destination ids, coordinates and size.

```
//Crop the original image to the specified rectangle (200x300)
//from the top-left corner (0, 0).
quickstart.crop_example_j(1, 4, 0, 0, 200, 300);
```

Rotating Images

In order to rotate an image we use the `ORDSYS.ORD_IMAGE.rotate()` procedure of the Oracle Multimedia PL/SQL package. The following code allows us to rotate an image by the angle specified and store the result in a new row.

Create the method `rotate_example_j()` to rotate the image in the `src_id` row, and store it into the `dst_id` row.

```
public void rotate_example_j(int src_id, int dst_id, float angle)
              throws SQLException
              {
        final String cropStmt  = "DECLARE "
                     + "    src_blob     BLOB;"
                     + "    dst_blob     BLOB;"
                     + "    src_id       NUMBER;"
                     + "    dst_id       NUMBER;"
                     + "BEGIN"
                     + "    src_id := ?;"
                     + "    dst_id := ?;"
                     + "    DELETE FROM image_blob_table WHERE id = dst_id;"
                     + "    INSERT INTO image_blob_table(id, image_blob) "
                     + "           VALUES (dst_id, empty_blob()) "
                     + "           RETURNING image_blob INTO dst_blob;"
                     + "    SELECT image_blob INTO src_blob FROM image_blob_table"
                     + "           WHERE id = src_id;"
                     + "    ORDSYS.ORD_IMAGE.rotate(src_blob,?, dst_blob);"
                     + "    UPDATE image_blob_table SET image_blob = dst_blob"
                     + "           WHERE id = dst_id; "
                     + "END;";

        final OracleCallableStatement pstmt =
                     (OracleCallableStatement)connection.prepareCall(cropStmt);

        //Binding the variables to the statement.
        pstmt.setInt(1, src_id);
        pstmt.setInt(2, dst_id);
        pstmt.setFloat(3, angle);

        //Execute the statement.
        pstmt.execute();
        connection.commit();
              }
```

Call the `rotate_example_j()` method passing the source and destination ids and the angle.

```
//Rotate the image 180 degrees.
```

```
quickstart.rotate_example_j(1, 5, 180);
```

Miscellaneous image processing operations

Now, we demonstrate the `ORDSYS.ORD_IMAGE.processCopy()` procedure of the Oracle Multimedia PL/SQL package which can be used to generate a new image by performing one or more image processing operations on a source image. The `processCopy()` procedure should be used when there is no specific procedure for performing the desired operation. For a full list of image processing operations refer to the *Oracle Multimedia Reference* Appendix D.

Create the method `processCopy_example_j()` to create a derivative image from a source image stored in the specified `src_id` row by performing the provided processing operations and writing the resulting image into the `dst_id` row. The original image remains unchanged.

```
public void processCopy_example_j(int src_id, int dst_id, String command)
             throws SQLException
             {
       final String cropStmt  = "DECLARE "
                    + "    src_blob     BLOB;"
                    + "    dst_blob     BLOB;"
                    + "    src_id       NUMBER;"
                    + "    dst_id       NUMBER;"
                    + "BEGIN"
                    + "    src_id := ?;"
                    + "    dst_id := ?;"
                    + "    DELETE FROM image_blob_table WHERE id = dst_id;"
                    + "    INSERT INTO image_blob_table(id, image_blob) "
                    + "           VALUES (dst_id, empty_blob()) "
                    + "           RETURNING image_blob INTO dst_blob;"
                    + "    SELECT image_blob INTO src_blob FROM image_blob_table"
                    + "           WHERE id = src_id;"
                    + "    ORDSYS.ORD_IMAGE.PROCESSCOPY(src_blob,?, dst_blob);"
                    + "    UPDATE image_blob_table SET image_blob = dst_blob"
                    + "           WHERE id = dst_id; "
                    + "END;";

       final OracleCallableStatement pstmt =
                    (OracleCallableStatement)connection.prepareCall(cropStmt);

       //Binding the variables to the statement.
       pstmt.setInt(1, src_id);
       pstmt.setInt(2, dst_id);
       pstmt.setString(3, command);

       //Execute the statement.
       pstmt.execute();
       connection.commit();
             }
```

Call the `processCopy_example_j()` method passing the source and destination ids and the image processing operations.

```
//Use the processCopy method to create a new image using two
//image processing operations.
quickstart.processCopy_example_j(1, 6, "fileformat = jfif fixedscale= 75 100");
```

Applying a Watermark to an Image

The `ORDSYS.ORD_IMAGE.applywatermark` procedure of the Oracle Multimedia PL/SQL package applies image or text watermarks onto an image. The following example shows how to apply a text watermark to an image. We

will generate a new image containing the watermarked image, which is a copy of the source image with overlayed text.
Create the method `applyWatermark_example_j()` to apply a text watermark to the image in the `src_id` row, and store the resulting image into the `dst_id` row.

```
public void applyWatermark_example_j(int src_id, int dst_id, String text)
              throws SQLException
              {
       final String applyWatermarkStmt  = "DECLARE "
                     + "    src_blob     BLOB;"
                     + "    dst_blob     BLOB;"
                     + "    dst_id       NUMBER;"
                     + "    prop         ordsys.ord_str_list;"
                     + "    log          VARCHAR2(2000);"
                     + "BEGIN"
                     + "    dst_id := ?;"
                     + "    prop := ordsys.ord_str_list("
                     + "           'font_name=Times New Roman', 'font_size=80');"
                     + "    DELETE FROM image_blob_table WHERE id = dst_id;"
                     + "    INSERT INTO image_blob_table(id, image_blob) "
                     + "           VALUES (dst_id, empty_blob()) "
                     + "           RETURNING image_blob INTO dst_blob;"
                     + "    SELECT image_blob INTO src_blob FROM image_blob_table"
                     + "           WHERE id = ?;"
                     + "    ORDSYS.ORD_IMAGE.applywatermark(src_blob,"
                     + "           ?, dst_blob,log, prop);"
                     + "    UPDATE image_blob_table SET image_blob = dst_blob"
                     + "           WHERE id = dst_id; "
                     + "END;";

       final OracleCallableStatement pstmt =
              (OracleCallableStatement)connection.prepareCall(applyWatermarkStmt);

       //Binding the variables to the statement.
       pstmt.setInt(2, src_id);
       pstmt.setInt(1, dst_id);
       pstmt.setString(3, text);

       //Execute the statement.
       pstmt.execute();
       connection.commit();
              }
```

Call the `applyWatermark_example_j()` method passing the source and destination ids and the text.

```
//Apply text watermarks("ORACLE") onto the original image
quickstart.applyWatermark_example_j(1,7,"ORACLE");
```

Exporting Images

An image stored in a blob column can be exported into a local disk file with the following steps.
Create the method `readImageFromDatabase`() to retrieve the image blob from the `image_blob_table` and write it to the local file system.

```
public void readImageFromDatabase(int src_id, String fileName)
              throws SQLException, IOException
              {

       //Define the PL/SQL block to retrieve the image blob.
       final String selectStmt  = "DECLARE "
                     + "    dst_blob     BLOB;"
                     + "BEGIN"
                     + "    SELECT image_blob INTO dst_blob FROM image_blob_table"
                     + "           WHERE id = ?;"
```

```
                + "    ? := dst_blob;"
                + "END;";

        final OracleCallableStatement pstmt =
                (OracleCallableStatement)connection.prepareCall(selectStmt);

        //Binding the variables to the statement.
        pstmt.setInt(1, src_id);
        pstmt.registerOutParameter(2, OracleTypes.BLOB);
        pstmt.execute();

        //Get the blob.
        BLOB img_blob = pstmt.getBLOB(2);

        try{
               //Declare the destination file, input and output streams.
               File file = new File(fileName);
               FileOutputStream outStream = new FileOutputStream(file);

               long remaining = img_blob.length();
               int size = 1024*1024; // 1MB.
               byte[] buffer = new byte[size];
               long position =1;

               //Write to the outputStream until there is no more data.
               while(true)
               {
                      if (remaining < size)
                             size=(int) remaining;

                      buffer= img_blob.getBytes(position, size);

                      outStream.write(buffer,0, size);
                      outStream.flush();

                      position+=size;
                      remaining-=size;

                      if (remaining ==0 )
                             break;
               }

               //Close stream.
               outStream.close();

        } catch (IOException e) {
               throw new IOException("Unable to write to " + fileName);
        }
               }
```

Call the `readImageFromDatabase`() method passing the connection object ,the source id and destination file.

```
//Retrieve the image we just created to create a local file.
quickstart.readImageFromDatabase(8, "flowers_export.jpg");
```

Adding and getting metadata.

This section shows how to use the `ORDSYS.ORD_IMAGE.putMetadata()` procedure of the Oracle Multimedia PL/SQL package to embed XMP metadata into an image and how to read it out again using the `ORDSYS.ORD_IMAGE.getMetadata()` procedure.

Create the method `putMetadata_example_j()` to add metadata to the image in the `src_id` row, and store the result into the `dst_id` row.

```
public void putMetadata_example_j(int src_id, int dst_id, String metadata)
              throws SQLException
              {
       final String putMetadataStmt =    "DECLARE"
                     + "    src_blob      BLOB;"
                     + "    dst_blob      BLOB;"
                     + "    dst_id        NUMBER;  "
                     + "    xmlData              XMLType;"
                     + "BEGIN"
                     + "    dst_id:=?;"
                     + "    SELECT image_blob INTO src_blob FROM image_blob_table "
                     + "           WHERE id = ?;"
                     + "    DELETE FROM image_blob_table WHERE id=dst_id; "
                     + "    INSERT INTO image_blob_table(id, image_blob) "
                     + "           VALUES(dst_id, empty_blob())"
                     + "           RETURNING image_blob INTO dst_blob;"
                     + "    xmlData := "
                     + "           xmltype(?, 'http://xmlns.oracle.com/ord/meta/xmp');"
                     + "    ORDSYS.ORD_Image.putMetadata(src_blob, dst_blob, xmlData,"
                     + "           'xmp', 'utf-8'); "
                     + "    UPDATE image_blob_table SET image_blob = dst_blob"
                     + "           WHERE id = dst_id; "
                     + "END;";

       final OracleCallableStatement pstmt =
                     (OracleCallableStatement)connection.prepareCall(putMetadataStmt);

       //Binding the variables to the statement.
       pstmt.setInt(1, dst_id);
       pstmt.setInt(2, src_id);
       pstmt.setString(3, metadata);

       pstmt.execute();
       connection.commit();
              }
```

Call the `putMetadata_example_j()` method passing the source and destination ids and the metadata to embed into the image

```
//Embed XMP metadata into an image.
final String XMPmetadata =
       "<xmpMetadata xmlns=\"http://xmlns.oracle.com/ord/meta/xmp\"> "
       + "<rdf:RDF xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\""
       + " xmlns:dc=\"http://purl.org/dc/elements/1.1/\"> "
       + "<dc:rights> "
       + "<rdf:Alt> "
       + "<rdf:li xml:lang=\"en-us\"> "
       + "Oracle Corporation 2014"
       + "</rdf:li>"
       + "</rdf:Alt>"
       + "</dc:rights>"
       + "</rdf:RDF>"
       + "</xmpMetadata>";
quickstart.putMetadata_example_j(1, 8, XMPmetadata);
```

In order to verify that we have added metadata to the image, we will create the method `getMetadata_example_j()` to retrieve the metadata using the `ORDSYS.ORD_IMAGE.getMetadata()` procedure.

```
public String getMetadata_example_j(int src_id) throws SQLException
{
       //Define the PL/SQL block to change the file format.
       final String metadataStmt = "DECLARE"
                     + "    metav        XMLSequenceType;"
                     + "    tmp          VARCHAR(4000);"
```

```
                        + "    meta          VARCHAR(32767);"
                        + "    dest_blob     BLOB;"
                        + "    cursor        xmlToString(x XMLSequenceType) IS"
                        + "          SELECT value(list_of_values).getstringval() "
                        + "          metadata FROM table(x) list_of_values;"
                        + "BEGIN"
                        + "    SELECT t.image_blob INTO dest_blob FROM image_blob_table t"
                        + "          WHERE T.ID = ?;"
                        + "    metav := ORDSYS.ORDImage.getMetadata(dest_blob,'ALL');"
                        + "    OPEN xmlToString(metav);"
                        + "    LOOP"
                        + "          fetch xmlToString INTO tmp;"
                        + "          meta:= meta || tmp;"
                        + "          EXIT WHEN xmlToString%NOTFOUND;"
                        + "    END LOOP;"
                        + "    CLOSE xmlToString;"
                        + "    ? := meta;"
                        + "END;";

            final OracleCallableStatement pstmt =
                        (OracleCallableStatement)connection.prepareCall(metadataStmt);

            //Binding the variables to the statement.
            pstmt.setInt(1, src_id);
            pstmt.registerOutParameter(2, OracleTypes.VARCHAR);

            pstmt.execute();

            return pstmt.getString(2);
      }
```

Call the `getMetadata_example_j()` method passing the destination id of the image we just created.

```
      //Retrieve the metadata from the image we just created.
      String metadata  = quickstart.getMetadata_example_j(8);
      System.out.println(metadata);
```

The output generated would look like the following (formatted for readability):

```
<ordImageAttributes xmlns="http://xmlns.oracle.com/ord/meta/ordimage"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/ord/meta/ordimage
http://xmlns.oracle.com/ord/meta/ordimage"><height>1704</height><width>2272</width><contentLengt
h>1696076</contentLength><fileFormat>JFIF</fileFormat><contentFormat>24BITRGB</contentFormat><co
mpressionFormat>JPEG</compressionFormat><mimeType>image/jpeg</mimeType></ordImageAttributes><exi
fMetadata xmlns="http://xmlns.oracle.com/ord/meta/exif"
xsi:schemaLocation="http://xmlns.oracle.com/ord/meta/exif http://xmlns.oracle.com/ord/meta/exif"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <TiffIfd>
      <Make tag="271">Canon</Make>
      <Model tag="272">Canon PowerShot S400</Model>
      <Orientation tag="274">top left</Orientation>
      <XResolution tag="282">180.0</XResolution>
      <YResolution tag="283">180.0</YResolution>
      <ResolutionUnit tag="296">inches</ResolutionUnit>
      <DateTime tag="306">2003-09-17T16:02:15</DateTime>
      <YCbCrPositioning tag="531">centered</YCbCrPositioning>
   </TiffIfd>
   <ExifIfd tag="34665">
      <ExposureTime tag="33434">0.0025</ExposureTime>
      <FNumber tag="33437">7.1</FNumber>
```

```
      <ExifVersion tag="36864">0220</ExifVersion>
      <DateTimeOriginal tag="36867">2003-09-17T16:02:15</DateTimeOriginal>
      <DateTimeDigitized tag="36868">2003-09-17T16:02:15</DateTimeDigitized>
      <ComponentsConfiguration tag="37121">YCbCr</ComponentsConfiguration>
      <CompressedBitsPerPixel tag="37122">3.0</CompressedBitsPerPixel>
      <ShutterSpeedValue tag="37377">8.65625</ShutterSpeedValue>
      <ApertureValue tag="37378">5.65625</ApertureValue>
      <ExposureBiasValue tag="37380">-1.0</ExposureBiasValue>
      <MaxApertureValue tag="37381">2.96875</MaxApertureValue>
      <MeteringMode tag="37383">Pattern</MeteringMode>
      <Flash tag="37385">
         <Fired>Yes</Fired>
         <Return>No strobe return function</Return>
         <Mode>Compulsory firing</Mode>
         <Function>Yes</Function>
         <RedEyeReduction>No</RedEyeReduction>
      </Flash>
      <FocalLength tag="37386">7.40625</FocalLength>
      <FlashpixVersion tag="40960">0100</FlashpixVersion>
      <ColorSpace tag="40961">sRGB</ColorSpace>
      <PixelXDimension tag="40962">2272</PixelXDimension>
      <PixelYDimension tag="40963">1704</PixelYDimension>
      <FocalPlaneXResolution tag="41486">8114.2856</FocalPlaneXResolution>
      <FocalPlaneYResolution tag="41487">8114.2856</FocalPlaneYResolution>
      <FocalPlaneResolutionUnit tag="41488">inches</FocalPlaneResolutionUnit>
      <SensingMethod tag="41495">One-chip color area</SensingMethod>
      <FileSource tag="41728">DSC</FileSource>
      <CustomRendered tag="41985">Normal process</CustomRendered>
      <ExposureMode tag="41986">Manual exposure</ExposureMode>
      <WhiteBalance tag="41987">Auto</WhiteBalance>
      <DigitalZoomRatio tag="41988">1.0</DigitalZoomRatio>
      <SceneCaptureType tag="41990">Standard</SceneCaptureType>
   </ExifIfd>
   <InteroperabilityIfd tag="40965">
      <InteroperabilityIndex tag="1">R98</InteroperabilityIndex>
   </InteroperabilityIfd>
</exifMetadata>
<xmpMetadata xmlns="http://xmlns.oracle.com/ord/meta/xmp"
xsi:schemaLocation="http://xmlns.oracle.com/ord/meta/xmp http://xmlns.oracle.com/ord/meta/xmp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:rights>
         <rdf:Alt>
            <rdf:li xml:lang="en-us"> Oracle Corporation 2014</rdf:li>
         </rdf:Alt>
      </dc:rights>
   </rdf:RDF>
</xmpMetadata>
<xmpMetadata xmlns="http://xmlns.oracle.com/ord/meta/xmp"
xsi:schemaLocation="http://xmlns.oracle.com/ord/meta/xmp http://xmlns.oracle.com/ord/meta/xmp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:rights>
```

```
         <rdf:Alt>
            <rdf:li xml:lang="en-us"> Oracle Corporation 2014</rdf:li>
         </rdf:Alt>
      </dc:rights>
   </rdf:RDF>
</xmpMetadata>
```

Compiling and running

The java code sample provided with this quick start has to be compiled including the jar file mentioned above. In order to run the Java program is necessary to include the following parameters: connection String, row id in which the original image will be stored, source image path, a second row id in which to begin storing generated images, and the destination image name to which to export an image; e.g.

```
java quickstart_image "jdbc:oracle:thin:@localhost:1521:orcl" "1" "flowers.jpg" "2"
"flowers_export.jpg "
```

Cleaning Up

To restore your database to its original state, you need to remove all of the objects that were created in this Quick Start as shown in the following example. Replace user "`scott`" with your database username.

```
connect scott
Enter password: password

drop table image_blob_table;
```

Conclusion

Using Java to work with Oracle Multimedia's PL/SQL package, we have shown how to import images into the database, retrieve image metadata, perform basic image processing operations, and export images to the local file system. Starting from these examples you can easily build and deploy your own Java solutions.

Oracle Multimedia provides more functionality than is covered in this Quick Start. Refer to the following documentation for more information: `Oracle Multimedia Reference`, and `Oracle Multimedia User's Guide`. Additional examples and articles are available on the Oracle Multimedia web page on the Oracle Technology Network at http://www.oracle.com/technetwork/database/database-technologies/multimedia/overview/index.html