

Delivering Rich Media Java Applications on Oracle9i Release 2

*An Oracle White Paper
June 2002*

Delivering Rich Media Java Applications on Oracle9i

Executive Overview	3
Introduction.....	3
What is oracle9i <i>interMedia</i> ?	4
New in Oracle9i	5
Oracle <i>intermedia</i> Classes and JDBC.....	5
Using Oracle <i>interMedia</i> from Java	6
Using Oracle <i>interMedia</i> from Java - JDBC and Database Connection	6
Using Oracle <i>interMedia</i> from Java - Processing Images.....	7
Using Oracle <i>interMedia</i> from Java - Image Retrieval.....	8
Using Oracle <i>interMedia</i> from Java - Using JMF to Play a Video	9
Uploading and Retrieving Multimedia Data in a Web-Based Environment	10
Image Retrieval and Delivery to a Browser.....	11
Uploading from the Browser to the Database.....	14
Retrieving <i>interMedia</i> data Using Streaming Media Servers.....	16
Using a Java IDE	16
Conclusion	18

Delivering Rich Media Java Applications on Oracle9i

EXECUTIVE OVERVIEW

Rich media types such as images, video clips, sound and animation play an increasingly important role in customer attraction as e-business continues to evolve. But retrieval and manipulation of this unstructured data has not been easy as it has traditionally been stored in files. Now Oracle offers a flexible solution for handling rich data in a Java development environment with Oracle9i, *interMedia*, and JDeveloper and other Java utilities.

INTRODUCTION

As today's vehicle of choice for e-business, the internet has brought about an unprecedented growth in dynamic, media rich applications. Why? It's simple -- applications rich in images, animation, and streaming audio and video appeal to customers. Customers like the convenience of the web but will not buy unless they have relevant product information. Rich media enhances that product information, making it more realistic and appealing to the potential consumer. In addition, rich media attracts customers to a web site and keeps them there, reducing the competitive challenge.

In parallel, object-oriented techniques and the Java programming language have become the developer's choice for producing this new breed of applications. In the past, rich media was stored in flat files as traditional databases were not equipped to deal easily with these more complex data types. Now this is changing as databases evolve to meet the challenge of handling complex data. Also, organizations are starting to recognize the disadvantages of flat files systems – the lack of central management, synchronization, availability, scalability, and transaction versioning that are necessary for the dynamic nature of internet applications. In essence, new media-rich applications require the same robust data management and delivery services for text, documents, media, geographical, and time-based information that are available for more traditional data. Oracle9i meets these new rich media and Java language requirements via Oracle *interMedia*. Oracle9i also provides a complete toolkit to handle rich media in an integral fashion with Java applications making it even easier for developers to get started.

WHAT IS ORACLE9I INTERMEDIA?

Rich content has the same needs as traditional data and can benefit from the advanced data management services found in established relational databases.

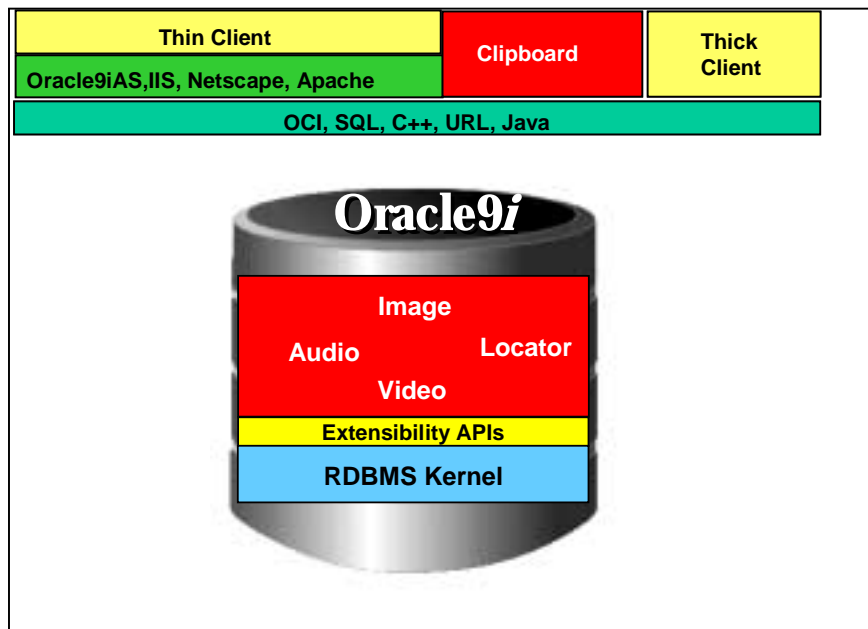
Oracle9i is an object-relational database management system that provides support for the definition of new object types and their integration with traditional business information, important features for today's e-business applications. Through the use of media datatypes, Oracle *interMedia* adds support that enables Oracle9i to manage and deliver image, audio, video, and geographical location information in an integrated fashion with other enterprise information. *interMedia* provides the means to add audio, image, and video columns or objects to existing database tables, insert and retrieve multimedia data, perform image processing on a number of popular image formats, and perform limited conversion between image formats.

The foundation for Oracle9i *interMedia* is the Oracle9i extensibility framework, a set of unique services that enables application developers to model complex logic and extend the core database services including optimization, indexing, type system, and SQL, to meet the specific needs of an application. Oracle has used these unique services to provide a consistent architecture for the rich datatypes supported by *interMedia*. *interMedia* uses object types, similar to Java or C++ classes, to describe multimedia data. These object types are called `ORDAudio`, `ORDImage`, `ORDVideo`, and `ORDDoc` and have attributes and methods associated with them

Oracle *interMedia* supports multimedia storage, retrieval, and management of:

- Binary large objects (BLOBs) stored locally in Oracle9i and containing audio, image, or video data
- File-based large objects, or BFILEs, stored locally in operating system-specific file systems and containing audio, image, or video data
- URLs containing audio, image, or video data stored on any HTTP server.
- Streaming audio or video data retrieved via specialized media streaming servers, such as RealNetworks.
- Any user-defined sources on other specialty servers.

interMedia is tightly integrated with SQL and the Oracle9i database engine, and is easily accessible through various thick and thin client interfaces including Java as indicated below.



New in Oracle9i

Through the use of a new relational interface introduced in Oracle9i, application designers who have chosen to store media data in BLOB columns can now use the full range of *interMedia* functionality, such as image processing, available in previous releases only through the use of the *interMedia* object types. Also introduced in Oracle9i is a new content-based retrieval feature, which allows images stored in a database to be searched using image matching technology. Given an image, content-based retrieval provides the ability to search images stored in a database table for other images using specific visual attributes such as color, texture, shape, and location. Examples of database applications where content-based retrieval is useful, that is, where the query is semantically of the form, "find objects that look like this one", include:

- Trademarks, copyrights, and logos
- Art galleries and museums
- Retailing
- Fashion and fabric design
- Interior design or decorating

ORACLE INTERMEDIA CLASSES AND JDBC

Oracle *interMedia* provides a set of Java classes (`OrdAudio`, `OrdImage`, `OrdVideo`, and `OrdDoc`) which allows applications to access and manipulate multimedia data stored in an Oracle9i database. Using these classes and JDBC, a developer can easily query the database, select rows from tables with media columns, upload and download media data between the database and application buffers or disk files, perform processing operations on the data, and output the data using a media player such as the one provided by Java Media Framework

(JMF). *interMedia* Java Classes allow JDBC result sets to include both traditional relational data and *interMedia* objects so that applications can easily select and operate on a result set containing *interMedia* columns and other relational data. Thus applications can retrieve *interMedia* objects in their Java programs using the standard JDBC `executeQuery` method. Through the use of *interMedia* Java classes, applications can access object attributes and invoke object methods. After modifying an object, the application updates the *interMedia* object in the database by executing a standard SQL UPDATE statement.

Using Oracle *interMedia* from Java

The following Java code samples demonstrate the ease of access to Oracle9i media via Java application logic. The database connection code is applicable to all Java and the database coding approaches while the upload and download between files and the use of JMF for playing audio and video is primarily applicable to thick clients or applets. The database table used in the image-based samples in this section consists of a unique integer identifier in column 1, and `ORDImage` objects in columns 2 and 3. The techniques used in these examples can be applied just as easily to more complex tables for such applications as parts assemblies, e-commerce catalogs, and web publications.

Using Oracle *interMedia* from Java - JDBC and Database Connection

The first example demonstrates loading the JDBC driver and establishing a connection to the database to be used by the subsequent examples. Note that this application connects to the `scott/tiger` sample schema in the default database. A production application could prompt the user for a schema name and password, or could read the schema name and password from a configuration file. These examples use images; however, the tasks being executed can be applied to audio and video data as well.

```
import java.io.*;
import java.sql.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;
import oracle.ord.im.*;

public class ImageDemo
{
    private String connectString = "jdbc:oracle:oci8:@";
    private String username = "scott";
    private String password = "tiger";

    private Connection conn = null;
    ...
    private void getConnection() throws SQLException
    {
        DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());
        conn =
DriverManager.getConnection(connectString,username,password);
    }
    ...
}
```

Using Oracle *interMedia* from Java - Processing Images

The second example shows how to load an image from a disk file into a database and how to process images in the database. Please note this section applies to images only and does not apply to audio and video objects. Two image processing activities occur:

- A GIF thumbnail of the image in the `image1` column is made and stored in the `image2` column.
- The image in the `image1` column is converted to the JFIF (JPG) format.

The application first disables auto-commit to ensure that the changes to the relational, object and BLOB data are all made in a single transaction, which is committed at the end of processing. The first SQL statement executed by the application inserts a new row containing two empty `ORDImage` database objects. The `ORDImage.init` method returns an `ORDImage` database object that is initialized with an empty BLOB. The application then executes a SQL `SELECT` statement to select the `ORDImage` database objects in the `image1` and `image2` columns in the newly-inserted row.

The `FOR UPDATE` clause in the `SELECT` statement allows the application to subsequently write to the BLOBs and update the `ORDImage` database objects in the selected row. Now note the methods called to enable the image conversion:

- The calls to the `getCustomDatum` method retrieve the `ORDImage` database objects from the result set and load them into the `OrdImage1` and `OrdImage2` Java objects for processing by the application.
- The call to the `loadDataFromFile` method reads the image data from the disk file and writes the data to the BLOB in the database.
- The call to the `setProperties` method parses the image data in the BLOB and loads the image attributes into the application's `OrdImage1` Java object.
- Using the call to the `processCopy` method, the application makes a GIF thumbnail of the image in `imageObj1`, storing the result in `imageObj2`.
- The call to the `process` method then converts the image in `imageObj1` to the JFIF (JPG) format.

Both the `processCopy` and `process` methods automatically update the image attributes in the application's `OrdImage` Java objects. The application then prepares a SQL `UPDATE` statement to update the `ORDImage` objects in the database from the contents of the application's `OrdImage` Java objects. The calls to the `setCustomDatum` method use the `imageObj1` and `imageObj2` Java objects to update the `image1` and `image2` columns in the database. Finally, the call to the `commit` method makes the changes to the database permanent.

```

private void doLoadAndProcess() throws SQLException,
IOException
{
    conn.setAutoCommit(false);

    Statement stmt1 = conn.createStatement();
    stmt1.executeUpdate("INSERT INTO imgdemo (id, image1,
image2) VALUES" +
        " (1, ORDSYS.ORDImage.init(),
ORDSYS.ORDImage.init())");

    OracleResultSet rs = (OracleResultSet)stmt1.executeQuery(
        "SELECT image1, image2 FROM imgdemo WHERE id = 1
FOR UPDATE");
    if (rs.next())
    {
        OrdImage imageObj1 = (OrdImage)rs.getCustomDatum(1,
OrdImage.getFactory());
        OrdImage imageObj2 = (OrdImage)rs.getCustomDatum(2,
OrdImage.getFactory());
        imageObj1.loadDataFromFile("image.bmp");
        imageObj1.setProperties();
        imageObj1.processCopy("maxScale=32 32,
fileFormat=GIF", imageObj2);
        imageObj1.process("fileFormat=JFIF");

        OraclePreparedStatement stmt2 =
            (OraclePreparedStatement)conn.prepareStatement(
                "UPDATE imgdemo SET image1 = ?, image2 = ?
WHERE id = 1");
        stmt2.setCustomDatum(1,imageObj1);
        stmt2.setCustomDatum(2,imageObj2);
        stmt2.execute();
        stmt2.close();
    }
    else
    {
        throw new SQLException("row not found");
    }

    rs.close();
    stmt1.close();
    conn.commit();
}

```

Using Oracle *interMedia* from Java - Image Retrieval

The third example retrieves one of the images processed by the previous example, displays the associated image meta data, and saves the image data to a disk file. As this example does not update the database, there is no need to disable auto-commit. The application executes a SQL SELECT to select the `ORDImage` database object in the `image2` column. A FOR UPDATE clause in the SELECT statement is not required as the application does not update the database. The call to the `getCustomDatum` method retrieves the `ORDImage` database object from the result set and loads it into the `OrdImage` Java object. The application then uses a series of calls to the `println` method to display a selection of attributes obtained from the image object using calls to various 'getter' methods. Finally, the application saves the image data to a disk file using the `getDataInFile` method.

```

private void doPropertiesAndSave() throws SQLException,
IOException
{
    conn.setAutoCommit(true);

    Statement stmt = conn.createStatement();
    OracleResultSet rs =
        (OracleResultSet)stmt.executeQuery("SELECT image2 FROM
imgdemo WHERE id = 1");
    if (rs.next())
    {
        OrdImage imageObj = (OrdImage)rs.getCustomDatum(1,
OrdImage.getFactory());
        System.out.println("");
        System.out.println("Format: " +
imageObj.getFormat());
        System.out.println("MimeType: " +
imageObj.getMimeType());
        System.out.println("Height: " +
imageObj.getHeight());
        System.out.println("Width: " +
imageObj.getWidth());
        System.out.println("ContentLength: " +
imageObj.getContentLength());
        System.out.println("ContentFormat: " +
imageObj.getContentFormat());
        System.out.println("CompressionFormat: " +
imageObj.getCompressionFormat());
        imageObj.getDataInFile("image.gif");
    }
    else
    {
        throw new SQLException( "row not found" );
    }

    rs.close();
    stmt.close();
}

```

The complete Java source for the preceding example can be found in the *interMedia* Java image demo directory,

<oracle_home>/ord/img/demo/java, of an Oracle9i installation, or in the samples section of the *interMedia* area on OTN at

<http://otn.oracle.com/products/intermedia>.

Using Oracle *interMedia* from Java - Using JMF to Play a Video

The final example in this section demonstrates the use of Java Media Framework (JMF) to play a video out of the database. The database table used in this example consists of a unique identifier in column 1 and an `ORDVideo` object in column 2.

This example uses Oracle *interMedia* Custom DataSource and DataSink classes to access the media in the database. Oracle *interMedia* Custom DataSource and DataSink classes are an extension to the current JMF version 2.0/2.1 developed by Sun Microsystems. This software allows a JMF application to upload and retrieve time-based media data stored in Oracle9i using *interMedia* `ORDAudio` and `ORDVideo` objects. Note that the Oracle *interMedia* Custom DataSource and DataSink classes must first be registered using the JMF Registry tool before they can be used by an application.

In the example, `MediaLocator` is the JMF class that is used to identify the source of the video in the database. The locator string format for Oracle *interMedia* Custom `DataSource` is as follows:

```
im://[<driverType>/]<jdbcConnectionString>/<user>:<password>/<queryString>
```

The `im` protocol indicates that Oracle *interMedia* Custom `DataSource` is to be used to retrieve the multimedia data. `<driverType>` is optional and can specify either an Oracle JDBC Thin driver or an Oracle JDBC OCI8 driver. The `<jdbcConnectionString>` format is based on the chosen JDBC driver. In the example, it specifies a host name, a port number and an Oracle database SID. `<user>:<password>` specifies the user name and password used to log in to the database. In the example, the `scott/tiger` sample schema is used. Finally, `<queryString>` specifies a SQL `SELECT` statement that is used to retrieve the multimedia data. The `SELECT` statement must retrieve a BLOB in the first column in the result set and the MIME type in the second column. `Player` is the JMF media player class while `DataSource` is the data sourcing class. Once a JMF data source and player have been created, playing the video is simply a matter of starting the player.

```
import javax.media.*;
import javax.media.control.TrackControl;
import javax.media.Format;
import javax.media.format.*;
import javax.media.protocol.*;
import javax.media.datasink.*;

MediaLocator locator = new
MediaLocator("im://thin/mynode:1521:orcl/scott:tiger/" +
              "SELECT
t.video.getcontent(), t.video.getMimeType()" +
              " FROM videostore t
WHERE videoid = 9");
DataSource source = Manager.createDataSource(locator);
Player player = Manager.createPlayer(source);

player.start();
```

UPLOADING AND RETRIEVING MULTIMEDIA DATA IN A WEB-BASED ENVIRONMENT

Oracle *interMedia* includes another Java component, called Oracle *interMedia* Java Classes for Java Servlets and JavaServer Pages, that facilitates the uploading and retrieval of multimedia data in a web-based environment. The examples in this section illustrate a simple photo album application implemented as a Java Servlet. However, the techniques illustrated in the examples can be applied easily to implement much more complex multimedia e-commerce applications that encompass additional *interMedia* technology, such as content-based retrieval using *interMedia* image matching technology.

To ensure that multiple requests can be processed simultaneously, the servlet assigns a dedicated JDBC connection to each request received by the `doGet` and

doPost methods. Using a dedicated JDBC connection for each request ensures that the servlet can deliver multiple items of multimedia data concurrently, and that multiple upload requests from different users each operate within the context of their own, independent transaction. In this servlet, a JDBC connection is assigned from a pool of connections when a new request is received and is returned to the pool at the end of each request. The following example shows how the doGet method assigns a JDBC connection to each request and how each request is processed by an instance of the PhotoAlbumRequest class. The actual processing performed by the doGet method is based on one of a number of request parameters, as shown in the example.

```
public void doGet( HttpServletRequest request,
HttpServletResponse response )
    throws ServletException, IOException
{
    Connection conn = null;
    try
    {
        conn = getConnection();
        PhotoAlbumRequest albumRequest = new PhotoAlbumRequest(
conn, request, response );

        String view_media = request.getParameter( "view_media"
);
        if ( view_media != null )
        {
            albumRequest.viewMedia( view_media );
        }
        else if ( request.getParameter( "view_entry" ) != null
)
        {
            albumRequest.viewEntry();
        }
        else
        {
            albumRequest.viewAlbum();
        }
    }
    catch ( SQLException e )
    {
        throw new ServletException( e.toString() );
    }
    finally
    {
        freeConnection( conn );
    }
}
}
```

The OrdHttpResponseHandler class facilitates the retrieval of multimedia data from an Oracle9i database and its delivery to a browser or other HTTP client from a Java Servlet. The OrdHttpJspResponseHandler class provides the same features for JavaServer Pages.

Image Retrieval and Delivery to a Browser

The next two examples in this section illustrates how to retrieve an image from an Oracle9i database and deliver it to a browser. The viewAlbum method in the PhotoAlbumRequest class is responsible for dynamically generating an HTML page that displays the contents of the photo album with thumbnail images

of each entry. Users click on the thumbnail image to view a full-size version of the photo. A screen-shot of the output from the `viewAlbum` method is shown below. The `viewMedia` method in the `PhotoAlbumRequest` class is responsible for actually delivering the image data, either a thumbnail or full-size image, to the browser. To complete the display functionality of the servlet, a third method, `viewPhoto`, is responsible for dynamically generating an HTML page that displays the full-size version of a photo.



The `viewAlbum` method, which is called by the servlet's `doGet` method, begins by obtaining the URI used to invoke the servlet. This is used to construct the full URL used to retrieve the thumbnail image, and to construct the URL in the anchor tag used to invoke the `viewPhoto` method to display the full-size image. The `viewAlbum` method then executes a SQL `SELECT` statement to select all the entries in the photo album. The tags to start an HTML table are then written to the servlet's `PrintWriter` output stream. For each entry in the album, the application calls the `getCustomDatum` method to retrieve the `ORDImage` database object from the result set and load it into the `OrdImage` `thumbNail` Java object.

The next statement displays the `id` and `description` of the entry and generates an HTML anchor tag with the following format

```
<a href="/<servlet-path>?view_entry=yes&id=<photo-id>">
   width=<width> >
</a>
```

Users click on the thumbnail image displayed by the image tag which sends the URL in the href attribute of the anchor tag to view an HTML page that displays the full-size photo. The height and width attributes of the image tag are obtained from the thumbNail object using calls to the getHeight and getWidth methods. The query string parameters in the URLs in the href and src attributes indicate to the servlet's doGet method the processing to be performed for a particular request.

```
void viewAlbum() throws ServletException, IOException,
SQLException
{
    String servletURL = request.getRequestURI();

    PreparedStatement stmt = conn.prepareStatement("SELECT
id,description,thumb FROM photos");
    OracleResultSet rset =
(OracleResultSet)stmt.executeQuery();

    out.println( "<p><table>" );
    out.println(
"<tr><th>ID</th><th>Description</th><th>Image</th></tr>" );
    while ( rset.next() )
    {
        OrdImage thumbNail = (OrdImage)rset.getCustomDatum( 3,
OrdImage.getFactory() );

        String id = rset.getString( 1 );
        String description = rset.getString( 2 );
        out.println( "<tr><td>" + id + "</td><td>" +
description + "</td>" +
" <td><a href=\" " + servletURL +
"?view=_entry=yes&id=" + id + "\">" +
" <img src=\" " + servletURL +
"?view_media=thumb&id=" + id + "\" +
" height=" + thumbNail.getHeight() + "
width=" + thumbNail.getWidth() +
" border=1></a></td></tr>" );
    }
    out.println( "</table></p>" );

    rset.close();
    stmt.close();
}
}
```

The viewMedia method is called by the servlet's doGet method to retrieve an image from the database and deliver it to the browser. The viewMedia method retrieves either the thumbnail or the full-size image based on the name of the column which is obtained by the doGet method from the view_media query string parameter. The viewMedia method starts by constructing a SQL SELECT statement to select the appropriate column from the row in the table with the specified id. An HTTP error is returned if the row is not found. The call to the getCustomDatum method retrieves the OrdImage database object from the result set and loads it into the OrdImage image Java object. The viewMedia method then creates an object of type OrdHttpServletResponse and calls the sendImage method to retrieve the image from the database and deliver it to the browser.

```

void viewMedia( String media ) throws ServletException,
IOException, SQLException
{
    PreparedStatement stmt =
        conn.prepareStatement( "SELECT " + media + " FROM
photos WHERE id = ?" );
    stmt.setString( 1, request.getParameter( "id" ) );
    OracleResultSet rset =
(OracleResultSet)stmt.executeQuery();
    if ( rset.next() )
    {
        OrdImage image = (OrdImage)rset.getCustomDatum(1,
OrdImage.getFactory());
        new OrdHttpServletResponse( request, response
).sendImage( image );
    }
    else
    {
        response.setStatus( response.SC_NOT_FOUND );
    }
    rset.close();
    stmt.close();
}
}

```

Uploading from the Browser to the Database

Form-based file uploading using HTML forms encodes form data and uploaded files in POST requests using the `multipart/form-data` format. The `OrdHttpUploadFormData` class facilitates the processing of such requests by parsing the POST data and making the contents of regular form fields and the contents of uploaded files readily accessible to a Java Servlet or JavaServer Page. The handling of uploaded files is facilitated by the `OrdHttpUploadFile` class, which provides an easy-to-use API that applications call to load image, audio, and video data into a database.

The final example in this section illustrates how to upload an image from a browser into an Oracle9i database. The `insertNewPhoto` method, called by the servlet's `doPost` method, first creates an instance of the `OrdHttpUploadFormData` class. The call to the `parseFormData` method parses the HTML form parameters from the `multipart/form-data` POST request. The application then gets the values of the `id` and `description` parameters as `String` objects, and the uploaded image file as an `OrdHttpUploadFile` object. The application uses a SQL `INSERT` statement, similar to the one illustrated in the previous section, to insert a new row into the database, followed by a SQL `SELECT` statement to select the `ORDImage` database objects in the newly-inserted row. The call to the `loadImage` method loads the uploaded photo into the `image` object. The call to the `processCopy` method then creates a thumbnail version of the image. The application then executes a SQL `UPDATE` statement to update the `ORDImage` objects in the `image` and `thumb` columns in the database from the contents of the `image` and `thumb` `OrdImage` Java objects in the application. Finally, the call to the `commit` method makes the changes to the database permanent.

```

void insertNewPhoto() throws ServletException, IOException,
SQLException
{
    OrdHttpUploadFormData formData = new OrdHttpUploadFormData(
request );
    formData.parseFormData();

    String id = formData.getParameter( "id" );
    String description = formData.getParameter( "description"
);
    OrdHttpUploadFile photo = formData.getFileParameter(
"photo" );

    conn.setAutoCommit( false );

    OraclePreparedStatement stmt =
        (OraclePreparedStatement)conn.prepareStatement(
            "INSERT INTO photos (id,description,image,thumb)
VALUES " +
            " ( ?, ?, ORDSYS.ORDImage.init(),
ORDSYS.ORDImage.init()" );
    stmt.setString( 1, id );
    stmt.setString( 2, description );
    stmt.executeUpdate();
    stmt.close();

    stmt = (OraclePreparedStatement)conn.prepareStatement(
        "SELECT image,thumb FROM photos WHERE id = ? FOR
UPDATE" );
    stmt.setString( 1, id );
    OracleResultSet rset =
(OracleResultSet)stmt.executeQuery();
    if ( !rset.next() )
    {
        throw new ServletException( "new row not found in
table" );
    }
    OrdImage image = (OrdImage)rset.getCustomDatum( 1,
OrdImage.getFactory());
    OrdImage thumb = (OrdImage)rset.getCustomDatum( 2,
OrdImage.getFactory());
    rset.close();
    stmt.close();

    photo.loadImage( image );
    image.processCopy( "maxScale=50,50", thumb );

    stmt = (OraclePreparedStatement)conn.prepareStatement(
        "UPDATE photos SET image = ?, thumb = ? WHERE
id = ?" );
    stmt.setCustomDatum( 1, image );
    stmt.setCustomDatum( 2, thumb );
    stmt.setString( 3, id );
    stmt.execute();
    stmt.close();

    conn.commit();
}

```

The complete Java source for the preceding example, together with a JSP example, can be found in the *interMedia* HTTP demo directory, `<oracle_home>/ord/http/demo/servlet` of an Oracle9i installation, or on the samples section of the *interMedia* area on OTN: <http://otn.oracle.com/products/intermedia>.

RETRIEVING *INTERMEDIA* DATA USING STREAMING MEDIA SERVERS

Oracle *interMedia* also provides support for some streaming audio and video formats, such as RealAudio and RealVideo, and the streaming servers that deliver them. Using a Java Servlet or JSP, the developer simply creates a URL whose protocol type indicates the streaming data format and whose virtual path and optional query string uniquely references the streaming content in the database.

Typically, a user would begin by querying the database for information, such as product or product-related information in an e-commerce application. If there is streaming audio or video available for some of the resulting query entries, the application indicates this to the user in some graphical means such as an anchor tag with an image on which the user can click, as illustrated in the previous example. If the user then chooses to play the audio or video, the URL associated with the anchor tag is dispatched to the appropriate browser plugin based upon the protocol field. At that point, the browser plugin establishes a network connection to the streaming server, and passes along the URL's contents. The URL uniquely identifies the database contents to be streamed back to the browser. The streaming server establishes a connection to the database via an Oracle plugin in the streaming server, retrieves streaming data from the database, and populates both the server side and the browser side plugin cache. When the caches are filled sufficiently, the browser plugin begins playing the media.

USING A JAVA IDE

Many software developers write their own Java code and will “cut and paste” from the previous examples. However, the use of Java integrated development environments (IDEs) is on a rapid increase. Why is there such interest in Java IDEs? Various surveys show that Java IDEs provide increased developer productivity, faster application development and easier maintenance of deployed applications.

Developers can now use a Java IDE, Oracle JDeveloper, to write Java applications using *interMedia* objects. JDeveloper is a visual component-based Java development environment tightly integrated with Oracle9i. With JDeveloper, it is easy to create thick and thin Java clients, servlets, JSPs, EJBs, and Java Beans -- and all of these can make use of the *interMedia* services for visually attractive application rich in multimedia content. A distinguishing feature of JDeveloper is Oracle Business Components for Java (BC4J), an application component framework that gives developers a set of intelligent software building-blocks to manage common facilities. The BC4J framework allows application developers to concentrate on the business logic and so deliver new functionality faster by extending and customizing application components using XML metadata, without requiring access to the underlying business component source code.

BC4J automatically recognizes the *interMedia* object types and integrates seamlessly with *interMedia* at multiple levels, thus providing great flexibility in the way in which developers can create new applications. For example, developers can choose

to create JSP applications with pre-packaged database manipulation components using the Business Components JSP Application Wizard. To create highly-customized JSP applications, developers can use the BC4J JSP Data Tag library. Finally, developers can create servlets or JSP applications, applets and stand-alone applications by programming directly against the *interMedia* BC4J domain classes. Each of these approaches is discussed below.

Business Components JSP Application Wizard allows users to generate a JSP application that is capable of querying, inserting, updating, and deleting data from the database. No programming at all is required to create a JSP application using the Business Components JSP Application Wizard – the wizard automatically integrates *interMedia* data objects along with standard relational data with no effort required of the developer. The JDeveloper generated JSP application uses BC4J JSP Data Tags for data accessing and content rendering. Users are able to view, insert, update and delete multimedia content just like handling other textual data by using the JSP application. The developers can debug or run the JSP application from inside the JDeveloper. When the development work is done, the developers can deploy the JSP application to an application server in the form of WAR file or EAR file from the JDeveloper.

To create JSP applications with highly customized user interfaces and application-specific database access functionality, developers can choose to use the BC4J Data Tag library. BC4J Data Tag library includes a set of BC4J *interMedia* Data Tags. These tags allow developers to write flexible media uploading/retrieving/rendering JSP code, such as uploading a video clip to the database or display an image taken from a database table. Using them together with other tags in BC4J Data Tag library, developers are able to create database bound, multimedia intensive JSP application with little efforts. The Component Palette is the place that BC4J Data Tag library resides. Developers use the Component Palette to drop JSP tags to their JSP pages to increase the productivity. The tag-based approach to build Business Components JSP web applications using the Data Tag library does not require extensive Java programming and is very much like coding an HTML page. In this way, developers have complete control over how the user interface is constructed and how the database is accessed.

If the developer is interested in is Java applet or standalone Java application instead of JSP application, the BC4J *interMedia* JClient Control component, *OrdMediaControl*, can be used to facilitate *interMedia* Java application development. BC4J JClient Controls are Swing controls that bind to Business Components data sources. The Model-View-Controller approach is used to build Java clients. *OrdMediaControl* takes advantage of the Java Media Framework to render image, audio and video media content. *OrdMediaControl* enables the user to insert, update, delete, retrieve, and view multimedia content stored in the *interMedia* objects in the database. To use *OrdMediaControl*, a developer simply needs to open the Java class in the JDeveloper UI Editor, then locate the

OrdMediaControl component in the Component Palette and drop it to the UI Editor.

Finally, developers can choose to program directly against the underlying *interMedia* BC4J Integration Package. This package is used internally by the Data Tag Library and the BC4J *interMedia* JClient Control. The *interMedia* BC4J Integration Package includes the *interMedia* BC4J domain classes and a set of utility classes. The `OrdImageDomain`, `OrdAudioDomain`, `OrdVideoDomain`, and `OrdDocDomain` *interMedia* BC4J domain classes are available with Release 9i of JDeveloper. These domain classes are wrappers of the *interMedia* Java Client classes described earlier and inherit all the underlying multimedia retrieval, upload, and manipulation methods. The *interMedia* Business Components domain classes support the `DomainOwnerInterface`, `LobInterface`, `AttributeList`, and `XMLDomainInterface` of the Business Components framework, and so provide built-in, integrated multimedia capabilities. The utility classes support the retrieval, rendering, and uploading of multimedia content. For example, any application can use the *interMedia* BC4J domain classes to facilitate uploading multimedia into the database. Servlet and JSP applications can use the `OrdURLBuilder` and `OrdPlayMedia` classes to build URLs and retrieve multimedia content from the database. `OrdURLBuilder` constructs URLs that locate *interMedia* object using the BC4J run-time framework, while `OrdPlayMedia` interprets the URLs to fetch the *interMedia* content from the database and deliver it to the browser.

CONCLUSION

Today, as we have shown, it's relatively easy for anyone to produce dynamic Java applications rich in media backed by a database. Object-relational database such as Oracle9i bring the advantages of scalability, availability, manageability and advanced search and query capabilities to complex datatypes. The new breed of databases such as Oracle9i now have rich media capabilities built in, reducing the pain for developers in their challenge to keep their organization's applications competitive. And this pain almost disappears with the use of an integrated Java IDE and application framework -- Oracle JDeveloper and Business Components for Java that can now create media-rich web applications as easily and quickly as more traditional ones. Because the addition of rich media enhances the user experience and attracts – and keeps – traffic at your web site, it is well worthwhile investigating this type of development solution.



Delivering Rich Media Java Applications on Oracle9i

May 2002

Author: Shirley Ann Stern

Contributing Authors: Simon Oxbury

Richard Wang

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

Oracle Corporation provides the software
that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.

Copyright © 2001 Oracle Corporation

All rights reserved.