
***interMedia* for Web Oriented Content Management: Getting Started**

Douglas Scherer
Core Paradigm

Abstract

Oracle8i introduces new functionality for asset management in its *interMedia* option. *interMedia* provides a set of services that strategically support work with content-rich data for Web-enabled applications. This paper provides an overview of *interMedia* to set you on the path to using it in your applications.

This paper will build towards a sample application built for uploading and retrieval of images via a browser. The application will use the DEMO account that is provided with most default Oracle databases. *interMedia* multimedia components support the storage, management, and manipulation of multimedia datatypes leading to their reuse (also called *re-purposing* in the digital assets disciplines) in a secure and recoverable environment. *interMedia* Text allows Oracle to be the central searchable repository for all types of data. The *interMedia* multimedia components can also interact with media servers such as the RealNetworks RealServer G2, which streams RealAudio and RealVideo to a browser or a stand-alone G2 client.

Introduction to *interMedia* Multimedia types

interMedia comes with three predefined object relational types, each of which stores the multimedia data as BLOBs, BFILEs, URLs, or streaming audio and video data from servers such as Oracle Video Server and the RealNetworks RealServer G2 Servers. The three predefined types, prefixed with ORD (ObjectRelationalDatatype), are shown below with example formats and file extensions:

- **ORDAudio** Supports the storage and management of audio data such as AIFF (.aif) and WAVE (.wav)
- **ORDImage** Supports the storage and management of image data such as GIFF (.gif) and TIFF (.tif)
- **ORDVideo** Supports the storage and management of video data such as AVI (.avi) and Quicktime (.mov)

Oracle *interMedia* Types Attributes

The *interMedia* multimedia datatypes are owned by the ORDSYS schema. All three types store data in the ORDSource object relational type, also owned by ORDSYS. Custom types can be defined that also store data in the ORDSource type. Each of the types contains methods and attributes that are detailed in the Oracle Document, *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*. The *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference* also contains lists of all of the *interMedia* native multimedia types (standard multimedia types that *interMedia* knows about). Table 1 shows the attributes of the ORDSource type. The attribute column shows the attribute name and its datatype.

Attribute	Description
LocalData BLOB	If the value is stored in a BLOB (internal/local), it contains the value.
SrcType VARCHAR2(4000)	Type of data storage for nonlocal data. Valid values: "FILE" for BFILE, "HTTP" for HTTP Server, "<name>" for a user-defined data storage type (this is a string that is used to derive a PL/SQL package name)
SrcLocation VARCHAR2(4000)	Location of the data if non-local. For srcType of "FILE"—<DIR> or name of directory object. For "HTTP" - <SourceBase> or URL (for example the base portion of URL less the filename at the end). For "<name>"—<iden> or access string
SrcName VARCHAR2(4000)	Name of the data object. For srcType of "FILE"—name of the file. For "HTTP"—<Source> or name of the object (for example, the filename at the end of a URL). For "<name>"—name of the object
UpdateTime DATE	Last modified timestamp
Local NUMBER	Indicates whether the object is stored internally. 1 or NULL = BLOB, 0 = external source

Table 1: Attributes of the ORDSource Object-Relational Type

Table 2 shows the attributes of the ORDImage *interMedia* datatype that will be used in the sample application.

Attribute	Description
Source ORDSource	The ORDSource storing the data
Height INTEGER	ORDImage: height of image in pixels; ORDVideo: height of each frame
Width INTEGER	ORDImage: width of image in pixels; ORDVideo: width of each frame
ContentLength INTEGER	Size of the on-disk image file in bytes. This value is the same as the one returned by DBMS_LOB.GET_LENGTH.
FileFormat VARCHAR2(4000)	File type. Example: TIFF
ContentFormat VARCHAR2(4000)	Type of image. Example: 8-bit grayscale

Attribute	Description
Compre3ssionFormat VARCHAR2(4000)	Image compression format. Example: JPEG
MimeType VARCHAR2(4000)	mimeType of the stored data. Example for ORDImage—image/tiff.

Table 2: List of ORDImage Attributes

Note: PL/SQL is case insensitive. Mixed case is used in the object definitions for clarity only.

The types include methods for manipulating the data. The complete list of methods can be found in the Oracle document *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference*. Table 3 shows the methods that are used in this paper.

Method	Description
GetContent()	Returns the content of the local data
GetContentLength()	Returns the size of the image in bytes (contentLength)
GetMimeType()	Returns the mime type of the stored image (mimeType)
GetUpdateTime()	Returns the time the image object was last updated (source.updateTime)
SetProperties	Fills in the attributes for an image if it is one of the native image types

Table 3: List of ORDImage Methods Used in This Paper

Add an interMedia Type to DEMO's Product Table

The DEMO schema's product table will now be extended by the addition of a multimedia column, named product_img, to the product table. The product_img column will be used to hold an image of the product. In its simplest form, the DDL to add this column is as follows:

```
ALTER TABLE product
  ADD(product_img ORDSYS.ORDImage);
```

Caution: The DEMO account is used as an example. You should make a backup of the DEMO account, or use an alternate account prior to experimenting with these techniques.

After the ALTER TABLE statement is executed, a DESCRIBE in SQL*Plus will show:

Name	Null?	Type
PRODUCT_ID	NOT NULL	NUMBER(6)
DESCRIPTION		VARCHAR2(30)
PRODUCT_IMG		ORDSYS.ORDIMAGE

Defining the Storage for *interMedia* Multimedia Object Types

Oracles's *interMedia* multimedia types can store data in BLOBs, in BFILES, or as a reference to a URL. The storage choice can be made for each instance of the object type. In other words, the storage choice for the column's value can be made on a row-by-row basis. When you are defining the product_img column as an ORDSYS.ORDImage (or any of the other multimedia types), a LOB segment is created, whether or not the value of the column will be stored internally.

LOB Storage Overview

LOBs work by storing a LOB locator with the table data (in the table's segment and in the same block as the row). The LOB locator is a pointer to the LOB data, which can be stored either *in-line* with the table's data or *out-of-line* from the table data (where the LOB is stored in its own segment). Whether in-line or out-of-line, when a LOB's data or *value* is stored in a segment, it is considered to be an *internal LOB*. The values of *External LOBs* reside outside the database as operating system files and so can never be stored in-line. BFILE is the only external LOB type. A LOB index is created for the LOB data in the same tablespace as the LOB data. LOB storage information can be seen in the USER_LOBS view.

Both internal LOBs (BLOB, CLOB, NCLOB) and external LOBs (BFILES) can store up to 4GB of data. By setting the storage parameters of the LOB correctly, you can allow both types to perform nearly equally as efficient in performance. Which of the two is appropriate for use in a particular application? The following rules of thumb can be used to help make this decision:

An internal LOB's advantage is that it receives the same benefits from an ORDBMS, such as security, manageability, backup and recovery, and transaction control as do standard datatypes. Using the issue of transaction control as an example, if a row is DELETED from the promotion table, the corresponding Word document is removed with it when the promotion column is defined as BLOB; it is not removed when the column is defined as a BFILE.

A BFILE, which at its simplest is a pointer to a file outside the database, will be used under one of two special conditions. First, it can be used to promote the inclusion of legacy data that is already stored as a set of documents in the filesystem. BFILES can be used to quickly make those files accessible to an Oracle database. Second, BFILES can be employed when there are applications that cannot connect to the database but need access to the documents. In this way, the database can share the document's data with those applications.

Internal LOB Default Value

LOBs need to have the LOB locator initialized before they can be used. If the data will be stored internally, the `localData` attribute can be given a default value. In the ALTER TABLE statement shown that follows, the `product_img` column will be defaulted to an `ORDImage` type that contains an `EMPTY_BLOB()` when no data is provided for this column at INSERT time. Notice that the `EMPTY_BLOB()` function is in the `localData` attribute position, `SYSDATE` is in the `updateTime` attribute position, and `1` is in the `local` position, signifying that the data is stored internally. The following ALTER TABLE statement shows how the `product_img` column can be added to the `product` table with storage parameters specified.

```
ALTER TABLE product
  ADD(product_img ORDSYS.ORDImage
    DEFAULT ORDSYS.ORDImage
      (ORDSYS.ORDSource(EMPTY_BLOB(),NULL, NULL, NULL,
        SYSDATE, 1
      ),
      NULL, NULL, NULL, NULL, NULL, NULL, NULL
    )
  )
  LOB(product_img.source.localdata)
  STORE AS product_img_seg
  (TABLESPACE user_images
    STORAGE(INITIAL 100K NEXT 100K PCTINCREASE 0)
    CHUNK 50K NOCACHE NOLOGGING ENABLE STORAGE IN ROW
  );
```

interMedia Web Agent Overview

The *interMedia* Web Agent works with Web servers to provide HTTP access to *interMedia* types managed by the database. The *interMedia* Web Agent performs retrieval and manipulation of multimedia data managed by the Oracle server on behalf of HTTP clients. It works with most Web/Application servers, including Oracle Application Server (OAS), Netscape Enterprise Server, Netscape FastTrack Server, and Microsoft Internet Information Server. The *interMedia* Web Agent is optimized for use with multimedia data and is multithreaded so that it will automatically scale to accommodate concurrent requests.

The *interMedia* Web Agent uses Database Agents for access to the database. A *Database Agent* contains the description of how the database will be accessed. It specifies, for example, the name of the database user to connect as; whether to prompt the user for that information when a request is made; and whether the *interMedia* Web Agent can be used to perform uploads, only to retrieve data, or to both manipulate and retrieve data from the database. The *interMedia* Web Agent, the admin Web Agent (a Web Agent used for administration of the *interMedia* Web Agent), and the Database Agent all keep their configuration information in the same file called `ORACLE_HOME/ord/web/admin/wsc.cfg`. For detailed configuration and installation information, see the Oracle document *Using Oracle8i interMedia with the Web*.

The focus of the remaining sections will be on building your own application using the *interMedia* Web Agent with the PL/SQL cartridge. It is worthwhile to note, though, that the *interMedia* Web Agent has a companion client side tool called the *interMedia* Clipboard. The Clipboard provides an interface for directly inserting, retrieving, and editing multimedia types.

Tip: As part of its work, the *interMedia* Clipboard generates procedures similar, but more complex than the ones shown in this chapter, in support of its work with the *interMedia* Web Agent. A good way to learn more about the *interMedia* Web Agent, the Clipboard, building your own applications, and even working with object types, is to review the generated Clipboard procedures. The Clipboard generated procedures can also be used as templates to your custom procedures.

The *interMedia* Web Agent and URLs

URLs addressed to the *interMedia* Web Agent include a request to perform either a mediaput (uploads to the database) or a mediaget (retrieval of multimedia data from the database). Both mediaputs and mediagets perform COMMITs and ROLLBACKs against the database. For debugging of Gets, ~mediagets can be used. A ~mediaget functions the same as a mediaget, except that it returns additional error related information when there is a processing problem. For debugging of Puts, appmediaput can be used. It functions the same way as a mediaput, except that an appmediaput returns error information in a format that is easier to use for non-browser applications (such as ones written in C++ or Java). Both Puts and Gets are implemented with the support of PL/SQL procedures.

The default virtual directory assigned to the *interMedia* Web Agent is *intermedia*. Thus, a URL that retrieves multimedia data might appear in the format shown following. In this example, a request is being made to an OAS listener at a host called *cesaria* on port 80. The virtual directory is *intermedia*, which has been mapped in the OAS to the *interMedia* Web Agent. Next, the Database Agent is specified as *my_database_agent*. A *mediaget* request is being made to retrieve from the database. The procedure, *my_get_procedure*, is being used by the *interMedia* Web Agent to retrieve the data in accordance with the information provided in the last position. This information is called the *path information*, and it includes the key value with which you can find the correct row in a table storing the data. Thus, in this example, data from the product table for *product_id* 105123 will be retrieved. If this URL were entered simply as an address in a browser and the data to be returned were an image, the image would be brought back as the sole item in the browser window.

```
http://cesaria:80/intermedia/my_database_agent/mediaget/my_get_procedure/105123
```

The URL can also be used as an image source on a more complex HTML page by including the URL in an tag as shown following:

```
<IMG SRC = "http://cesaria:80/intermedia/my_database_agent/mediaget/my_get_procedure/105123">
```

Tip: It is important to remember that the *interMedia* Web Agent can work with any of the *interMedia* types. In fact, once you have command over creating the *mediaget* and *mediaput* procedures that support the *interMedia* Web Agent calls to the database, it is possible to use the *interMedia* Web Agent for multimedia types of your own definition. The *interMedia* Web Agent can act upon CLOBs, BLOBs, and VARCHAR2s in addition to the multimedia types.

Creating a Put Procedure to Perform *interMedia* Web Agent Transactions

A Put procedure is the first one called by the *interMedia* Web Agent during a *mediaput* request. The main job of the Put procedure is to pass back the LOB locator if the value is stored internally and NULL for the LOB locator if the value is stored as a BFILE, URL or user-defined data storage type. Table 4 shows a list of parameters used in Put procedures for multimedia data that is stored as a BLOB. For the full list of parameters see the Oracle Document *Using Oracle8i interMedia with the Web*.

Parameter Name	Datatype	Required	Description
ord_content_blob	OUT BLOB	Y	Returns the LOB locator
ord_content_path	IN VARCHAR2	Y	Contains the identifying key value for the data
ord_content_type	OUT VARCHAR2	N	Specifies the mimetype of the multimedia data
http_status	OUT INTEGER	N	Returns an <i>interMedia</i> Web Agent predefined status code, based on standard HTTP status codes. Code Ranges: 200–299: success; 300–399: redirection and cache; 400–499: client-side error; 500–599: server-side error.
http_redirect	OUT VARCHAR2	N	Returns the URL to be used instead of what the user originally requested

Table 4: Partial List of Parameters of a *interMedia* Web Agent Put Procedure

Note: The parameters prefixed with http_ are particular to the interMedia Web Agent. When used in the procedures, they must have the names specified in Table 4.

The procedure `wa_product_img_put` (shown below), for which the main purpose is to pass a valid LOB locator to the Web Agent, will be used to UPDATE the `product_img` column. This procedure takes two parameters:

- **ord_procedure_path** Contains a `product_id`
- **ord_content_blob** Contains the LOB locator that the *interMedia* Web Agent will use to populate the BLOB data

```
CREATE OR REPLACE PROCEDURE wa_product_img_put
(
  ord_procedure_path IN VARCHAR2,
  ord_content_blob OUT BLOB
)
AS
BEGIN
  SELECT p.product_img.getContent()
  INTO ord_content_blob
  FROM product p
  WHERE product_id = ord_procedure_path
  FOR UPDATE;
END;
```

Creating a Set Procedure to Handle *interMedia* Web Agent Transactions

The Set procedure populates the attributes of the `ORDSYS.ORDImage` column so that they are synchronized with the data value. For example, if a TIFF that was contained in the `product_img` column was subsequently UPDATED to contain a JPEG, the Set procedure would make sure that the change was reflected in the object. The Set procedure does this by calling the `ORDImage.setProperties` method.

After the Set procedure runs, the *interMedia* Web Agent typically returns the success message shown in Figure 1 to the browser.



Figure 1: The default *interMedia* Web Agent success page

The procedure `wa_product_img_set`, that follows, uses the `http_redirect` parameter to return, instead, an application-specific success message. The `http_redirect` parameter is assigned the value of a URL that makes use of the `product_img_app.head` procedure (shown in the below section "*Code for the Product_img_app Application.*") Upon success, the requestor will see the page shown in Figure 2. The success page is the result of calling through use of the *interMedia* Web Agent to upload an image for product 100860.

```
CREATE OR REPLACE PROCEDURE wa_product_img_set
(
  ord_procedure_path VARCHAR2,
  http_redirect OUT VARCHAR2
)
AS
  ob_product_img product.product_img%TYPE;
  v_rowid UROWID;
BEGIN
  SELECT ROWID, product_img
    INTO v_rowid, ob_product_img
    FROM product
   WHERE product_id = ord_procedure_path
   FOR UPDATE;

  ob_product_img.setproperties();

  UPDATE product
     SET product_img = ob_product_img
    WHERE ROWID = v_rowid;

  http_redirect := 'http://natalie:80/demo/^head' ||
                  '?i_title=DEMO's+product+image+screen:+success.' ||
                  '&i_head=The+image+for+product+id+' ||
                  ord_procedure_path || '+has+been+updated+successfully.';
END;
```

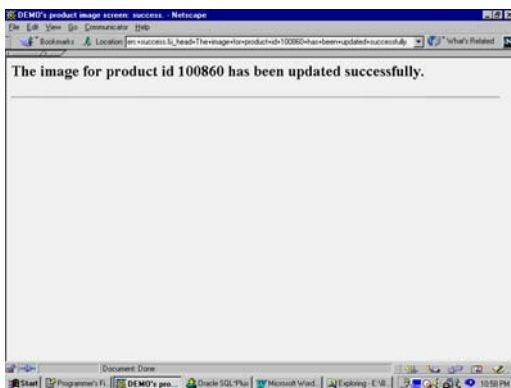


Figure 2: A custom success screen

Creating a Get Procedure to Handle Retrieval of Multimedia Data

The Get procedure retrieves the multimedia object from the database. Using the `wa_product_img_get` procedure shown following, it will retrieve the image of a product stored in `product_img` column. The `wa_product_img` procedure takes seven parameters. The `ord_procedure_path` parameter contains the `product_id`, while the three parameters that begin with `ord_content_pass` out information about the object. The three parameters beginning with `http_` are used to determine the status of that object in the client's cache. If the most recent version of the object is in the cache, the object data does not need to be retrieved.

```
CREATE OR REPLACE PROCEDURE wa_product_img_get
(
  ord_procedure_path VARCHAR2,
  ord_content_type OUT VARCHAR2,
  ord_content_length OUT NUMBER,
  ord_content_blob OUT BLOB,
  http_if_modified_since IN VARCHAR2,
  http_status OUT VARCHAR2,
  http_last_modified OUT VARCHAR2
)
AS
  ob_product_img ORDSYS.ORDIMAGE;
BEGIN
  SELECT product_img
  INTO ob_product_img
  FROM product
  WHERE product_id = ord_procedure_path;
  http_status := ORDWEBUTL.CACHE_STATUS
  (ob_product_img.getupdatetime(), http_if_modified_since,
   http_last_modified
  );
  IF http_status != 304
  THEN
    -- The client's cache does not contain the most
    -- recent data, the cache must be reloaded. If the
    -- cache were up-to-date the program could return
    -- immediately.
    ord_content_type := ob_product_img.getmimetype();
    ord_content_length := ob_product_img.getcontentlength();
    ord_content_blob := ob_product_img.getcontent();
  END IF;
END;
```

Creating an Application that Uses the *interMedia* Web Agent with the PL/SQL Cartridge

Applications using the PL/SQL cartridge can be created to upload and retrieve the multimedia data managed by the database. In this section, two simple applications will be created to work with the images of products held in the `product_img` column. These procedures will be contained in a package called `product_imp_app`.

Uploading the Product Image

The HTML form used to upload an image using the *interMedia* Web Agent must contain the specification for the action that calls the *interMedia* Web Agent's Put procedure; the variable `ord_post_put_call`, which contains the name of the Set procedure; `ord_procedure_path`, which contains the key value (for example, the product id input by the user); and `ord_content`, which contains the specification of the file to be uploaded. The HTML to produce the form for the sample application is shown following. The method of the form action must be `post`, and the enctype must be `multipart/form-data`. For these examples, a Database Agent named `demo_da` will be used along with an OAS application named `demo`.

```
<HTML>
<HEAD>
<TITLE>DEMO's Upload product image screen</TITLE>
<H2>Upload a Product Image</H2>
<HR>
</HEAD>
<BODY>
<FORM
  ACTION="http://cesaria:80/intermedia/demo_da/mediaput/wa_product_img_put"
  METHOD="POST" ENCTYPE="multipart/form-data"
>
<BR>
<B>Product ID: </B>
<INPUT TYPE="text" NAME="ord_procedure_path" SIZE="6">
<INPUT TYPE="hidden" NAME="ord_post_put_call"
  VALUE="wa_product_img_set"
>
<B>Product Image: </B>
<INPUT TYPE="file" NAME="ord_content" size=35>
<BR>
<BR>
<INPUT TYPE="submit" VALUE="Upload Image Now">
<INPUT TYPE="reset" VALUE="Reset">
</FORM>
</BODY>
</HTML>
```

The `product_img_app` package header will contain a call to the procedure `upload_product_img` shown following. `Upload_product_img` will contain calls to the PL/SQL Web development toolkit to produce a form similar to the one preceding. The `product_img_app.upload_product_img` procedure can be retrieved as a Web page with the following URL:

`http://cesaria:80/demo/^product_imp_app.upload_product_img`

Once the page is retrieved, it will contain two fields: one to enter the `product_id` and the second specifying the file from the local filesystem to upload in the `product_img` column. Figure 3 shows the page with data entered into the fields.

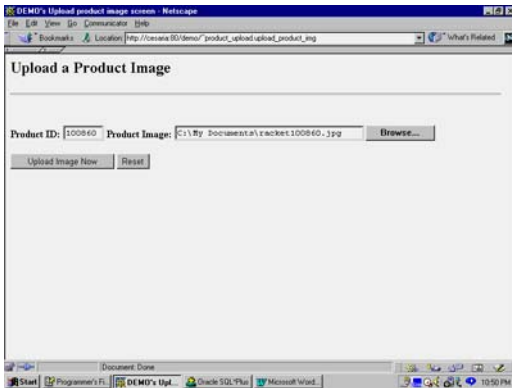


Figure 3: Page produced by `product_imp_app.upload_product_img`

When the Upload Image Now button is pressed and the image is successfully uploaded, the screen shown in Figure 2 will appear.

Viewing the Product Image

Once the image has been stored in the database, PL/SQL procedures can retrieve it via the *interMedia* Web Agent. Two procedures will be added to the `product_img_app` package. `Review_product_img` will show a field in a browser that accepts a `product_id` for which the user wants to see the product image. `Retrieve_product_img` will return a Web page containing the product's id and description along with an image of that product. `Retrieve_product_img` takes one parameter – `i_product_id`—which will contain the id of the product to display. Additions to the code are shown in bold.

The `product_img_app.review_product_img` procedure is called with the following URL:

```
http://cesaria:80/demo/^product_imp_app.review_product_img
```

It brings up the page shown in Figure 12-8. The figure shows that a search will be performed for product 100860's image.

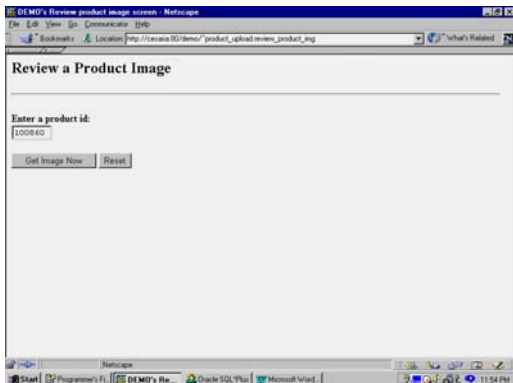


Figure 4: The review product image page

Figure 5 shows the page retrieved for the product, with the description included in the header and with the image pulled from the database via the *interMedia* Web Agent.

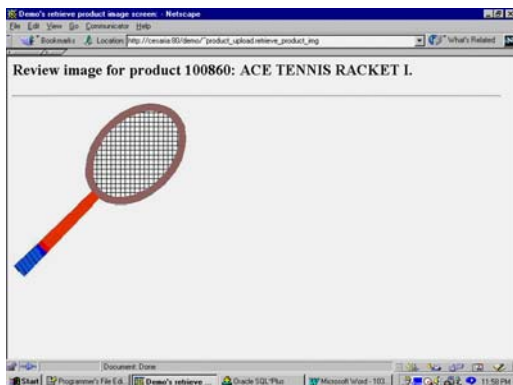


Figure 5: Retrieve product image page

Code for the `Product_img_app` Application

```
CREATE OR REPLACE PACKAGE product_img_app
AS
  PROCEDURE head
    (i_title VARCHAR2,
     i_head VARCHAR2,
     i_close BOOLEAN DEFAULT FALSE
    );
  PROCEDURE upload_product_img;
```

```

PROCEDURE review_product_img;
PROCEDURE retrieve_product_img
    (i_product_id product.product_id%TYPE);
END;

CREATE OR REPLACE PACKAGE BODY product_img_app
AS
    -- head will create a standard looking title and
    -- heading for each screen in the application.
    PROCEDURE head
        (i_title VARCHAR2,
         i_head VARCHAR2,
         i_close BOOLEAN DEFAULT FALSE
        )
    IS
    BEGIN
        HTP.HTMLOPEN;
        HTP.HEADOPEN;
        HTP.TITLE(i_title);
        HTP.PRINT('<H2>' || i_head || '</H2>');
        HTP.HR;
        HTP.HEADCLOSE;
        IF i_close
        THEN
            -- Close the HTML page
            HTP.HTMLCLOSE;
        END IF;
    END;
    PROCEDURE upload_product_img
    IS
    BEGIN
        -- Put an informational header on the browser screen.
        head('DEMO's Upload product image screen',
            'Upload a Product Image'
        );
        HTP.BODYOPEN;
        -- Create an HTML form to accept a product image.
        HTP.FORMOPEN('http://cesaria:80/intermedia/demo_da/' ||
            'mediaput/wa_product_img_put', 'POST',
            cenctype => 'multipart/form-data'
        );
    END;
END;

```

```

        );

    HTP.BR;

    HTP.PRINT('<B>Product ID: </B>');

    HTP.FORMTEXT('ord_procedure_path', 6);

    HTP.FORMHIDDEN('ord_post_put_call', 'wa_product_img_set');

    HTP.PRINT('<B>Product Image: </B>');

    HTP.FORMFILE('ord_content', cattributes => 'size=35');

    HTP.BR;

    HTP.BR;

    HTP.FORMSUBMIT(NULL, 'Upload Image Now');

    HTP.FORMRESET;

    HTP.FORMCLOSE;

    HTP.BODYCLOSE;

    HTP.HTMLCLOSE;

END;

PROCEDURE review_product_img
IS
BEGIN
    -- Put an informational header on the browser screen.
    head('DEMO's Review product image screen',
        'Review a Product Image'
    );

    HTP.BODYOPEN;

    -- Create an HTML form to accept a product id.
    HTP.FORMOPEN('http://cesaria:80/demo/^product_imp_app.' ||
        'retrieve_product_img'
    );

    HTP.PRINT('<B>Enter a product id:</B>');

    HTP.BR;

    HTP.FORMTEXT('i_product_id', 6);

    HTP.BR;

    HTP.BR;

    HTP.FORMSUBMIT(NULL, 'Get Image Now');

    HTP.FORMRESET;

    HTP.FORMCLOSE;

    HTP.BODYCLOSE;

    HTP.HTMLCLOSE;

END;

PROCEDURE retrieve_product_img
(i_product_id product.product_id%TYPE)

```

```

IS
  v_description product.description%TYPE;
BEGIN
  SELECT description
    INTO v_description
   FROM product
  WHERE product_id = i_product_id;
  -- Put an informational header on the browser screen.
  head('Demo's retrieve product image screen:',
    'Review image for product ' ||
    i_product_id || ': ' || v_description || '.');
  -- Create an <IMG> tag that uses the interMedia
  -- Web Agent as the image source.
  HTP.IMG('http://cesaria:80/intermedia/demo_da/mediaget/' ||
    'wa_product_img_get/' || i_product_id
  );
  HTP.HTMLCLOSE;
EXCEPTION
  WHEN NO_DATA_FOUND
  THEN
    -- Provide a custom error message to the user when a product
    -- does not exist for the user-provided product_id.
    head('Demo's retrieve product image screen:' ||
      'failure', 'Product ID ' || i_product_id ||
      ' does not exist.', TRUE
    );
  WHEN OTHERS
  THEN
    -- Provide a custom error message to the user for all
    -- other errors.
    head('Demo's retrieve product image screen:' ||
      ' failure', 'Product ID ' || i_product_id ||
      ' retrieval failed with Oracle Error ' ||
      SQLCODE || '.', TRUE
    );
  HTP.HTMLCLOSE;
END;
END;

```

About the Author

Douglas Scherer is president of Core Paradigm, a firm providing consulting, mentoring and formal training solutions primarily for Oracle database application and management environments. He is a frequent speaker at conferences and user-group meetings internationally and has appeared in *Visions of the New Millennium*, a series seen on PBS and its affiliates. He is lead author of *Oracle 8i Tips & Techniques* and contributing editor to the *Oracle Designer Handbook*, second edition, both published by Osborne (Oracle Press). He also chairs the database track at Columbia University's Computer Technology and Applications program. He holds an M.S. from Columbia University. <http://www.coreparadigm.com>