Oracle8™ Image Cartridge

User's Guide

Release 8.0.4

November 1997

Part No. A55713-02

Oracle8 Image Cartridge is an integral component of Oracle Corporation's Network Computing Architecture.

ORACLE®

Enabling the Information Age™

Oracle8 Image Cartridge User's Guide

Part No. A55713-02

Release 8.0.4

Primary Author: Jeff Hebert

Contributors: Cathy Trezza, Susan Shepard

Contents

Send Us Your Comments

Oracle8 Image Cartridge User's Guide, Release 8.0.4

Part No. A55713-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available).

You can send comments to us in the following ways

- e-mail: nedc_doc@us.oracle.com
- FAX - 603.897.3269. Attn: Image Cartridge Documentation
- postal service
  Oracle Corporation
  Oracle Image Cartridge Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you would like a reply, please include your name, address, and telephone number.

Preface

This guide describes how to use Oracle8 Image Cartridge.

Image Cartridge requires Oracle8 Enterprise Edition. Oracle8 and Oracle8 Enterprise Edition have the same basic features. However, several advanced features, such as data cartridges, are available only with the Enterprise Edition, and some of these features are optional. For example, to extend Image Cartridge with new types, you must have the Enterprise Edition and the Objects Option.

For information about the differences between Oracle8 and the Oracle8 Enterprise Edition and the features and options that are available to you, see *Getting to Know Oracle8*.

Intended Audience

This guide is intended for anyone who wants to store, retrieve, and manipulate image data in an Oracle database, including developers of image specialization cartridges.

Structure

This guide contains three chapters and three appendixes:

Chapter 1	Introduces image and data cartridge concepts and provides an overview of the Image Cartridge functions.
Chapter 2	Provides examples of using the Image Cartridge types and methods.
Chapter 3	Describes the procedures you can use to manipulate image data.
Appendix A	Describes the supported image data formats.
Appendix B	Describes how to install Image Cartridge.

Appendix C Lists potential errors, their causes, and user actions to correct them.

Related Documents

> **Note:** For information added after the release of this guide, refer to the online README.TXT file in your ORACLE_HOME directory. Depending on your operating system, this file may be in:
>
> `ORACLE_HOME/ord/img/admin/README.TXT`
>
> Please see your operating-system specific installation guide for more information.

For more information about using this data cartridge in a development environment, see the following documents in the Release 8.0 Oracle Server documentation set:

- *Oracle Call Interface Programmer's Guide*
- *Oracle8 Application Developers Guide*
- *Oracle8 Concepts*
- *PL/SQL User's Guide and Reference*

Conventions

In this guide, Oracle8 Image Cartridge is sometimes referred to as Image Cartridge, or simply, *the cartridge*. Oracle Corporation's Network Computing Architecture is referred to as, *the architecture*.

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are also used in this guide:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.

Convention	Meaning
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1

Introduction

Oracle8 Image Cartridge is an extension to Oracle8 Enterprise Edition and provides image storage, retrieval, and format conversion capabilities through an object data type (ODT). This cartridge supports image storage using binary large objects (BLOBs) and references to image data residing in external files (BFILEs).

Image Cartridge is a building block for various imaging applications rather than being an end-user application in itself. It consists of ODTs along with related methods for managing and processing image data. Some example applications for this cartridge are:

- digital art galleries or museums
- real estate marketing
- document imaging
- stock photo collections (for example, for fashion designers or architects)

These applications have certain distinct requirements and some things in common. The purpose of the image ODTs is to support both. With Image Cartridge, images can be managed as easily as standard attribute data.

Image Cartridge provides the foundation for static, two-dimensional images in Oracle databases. The images may be bitonal (black and white) document images, grayscale photographs, or three-color photographs. The cartridge provides the means to add image columns to existing tables, insert and retrieve images, and convert between popular application formats. The cartridge also provides sufficient capabilities for extending existing application databases with images or building new end-user image database applications.

1.1 Image Concepts

This section contains information about digital images and using Image Cartridge to build either image applications or specialized image data cartridges.

1.1.1 Digital Images

Image Cartridge provides the mechanism to integrate the storage and retrieval of digital images in Oracle databases via the Oracle8 Server.

Image Cartridge supports two-dimensional, static, digital images stored as binary representations of real world objects or scenes. Images may be produced by a document or photograph scanner, a video source such as a camera or VCR connected to a video digitizer or frame grabber, other specialized image capture devices, or even by program algorithms. Capture devices take an analog or continuous signal such as the light that falls onto the film in a camera, and convert it into digital values on a two-dimensional grid of data points known as pixels. Devices involved in the capture and display of images are under application control.

1.1.2 Image Components

A digital image consists of attributes that describe the characteristics of the image, and the image data itself (the digitized bits). Occasionally, image applications will associate application-specific information with an image, such as including the name of the person pictured in a photograph.

The minimal attributes of an image may include such things as its size (height in scan lines and width in pixels), the resolution at which it was sampled, and the number of bits per pixel in each of the colors sampled.

The image data (pixels) can have varying depths (bits per pixel) depending on how the image was captured, and can be organized in various ways. The organization of the image data is known as the **data format**. Image Cartridge can read and write image data using a variety of the data formats in use today. See Appendix A, “File and Compression Formats” for a list of supported data formats.

The storage space required for digital images can be large compared to traditional attribute data such as numbers and text. Many compression schemes are available to squeeze an image into fewer bytes, thus reducing storage device and network load. **Lossless compression** schemes squeeze an image so that when it is decompressed, the resulting image is bit-for-bit identical with the original. **Lossy compression** schemes do not result in an identical image when decompressed, but rather, one in which the changes may be imperceptible to the human eye.

1.1.3 Interchange Formats

Image **interchange format** describes a well-defined organization and use of image attributes, data, and often compression schemes, allowing different applications to create, exchange, and use images. Interchange formats are often stored in or as disk files. They may also be exchanged in a sequential fashion over a network and be referred to as a protocol. There are many application subdomains within the digital imaging world and many applications that create or utilize digital images within these. Image Cartridge supports many interchange formats (see Appendix A, "File and Compression Formats").

1.1.4 Data Cartridges

Within Oracle Corporation's Network Computing Architecture, **data cartridges** facilitate the storage and retrieval of complex data types required by nontraditional database applications such as geographic information systems, imaging, workflow, document management, and digital libraries. These applications are built using components or modules that support the capture, input, processing, analysis, storage, retrieval, and display of the complex data types.

A data cartridge is the mechanism by which clients, application-specific servers, and database servers can be easily and reliably extended. Image Cartridge plugs into Oracle® Universal Server and provides support for image domain-specific types, methods, and interfaces. Image Cartridge focuses on a set of image data representation and access mechanisms sufficient to support many image applications, and the development of more specialized image cartridges.

The primary focus of Image Cartridge is to provide storage, retrieval, reformatting, compression, decompression, and a minimal set of data manipulation services including scaling and cropping.

1.2 Object Relational Technology

Oracle Universal Server is an **object relational** database management system. This means that in addition to its traditional role in the safe and efficient management of relational data, it now provides support for the definition of object types including the data stored in an object and the operations (methods) that can be performed on that object. This mechanism, combined with integral support for binary large objects (BLOBs), provides the basis for adding complex objects such as digital images to Oracle databases.

1.2.1 Storing Images

Image Cartridge is capable of storing digital images within the Oracle database under transactional control through the BLOB mechanism. It is also capable of externally referencing digital images stored in flat files. While this later mechanism is particularly convenient for integrating pre-existing sets of flat file images with an Oracle database, these images will not be under transactional control. BLOB stored images have an object relational type known as **ORDImgB**, while BFILE stored images have an object relational type known as **ORDImgF**.

BLOBs are stored in the database tablespaces in a way that optimizes space and provides efficient access. BLOBs may or may not stored inline with other row data. Depending on the size of the BLOB, a locator is stored in the row and the actual BLOB (up to 4 gigabytes) is may be stored in other tablespace. The locator can be considered a pointer to the actual location of the BLOB value. When you select a BLOB, you are selecting the locator instead of the value, although this is done transparently. An advantage of this design is that multiple BLOB locators can exist in a single row. For example, you might want to store both an employee's photograph and his or her thumbprint or retina scan for security purposes.

BFILEs are not under the database's transactional control. Only the locator for a BFILE image is stored in the row. See the *Oracle8 Application Developer's Guide* and the *Programmer's Guide to the Oracle Call Interface* for detailed information on using BLOBs and BFILEs.

1.2.2 Querying Images

Once stored within an Oracle database, an image can be queried and retrieved by finding a row in a table that contains the image using the various alphanumeric columns (attributes) of the table. An example would be, selecting a photo from the EMPLOYEE table where the employee name is Jane Doe.

The collection of digital images in the database must be related to some set of attributes or keywords that describe the associated content. The image content can be described with textual components and numeric attributes such as dates and identification numbers. For Oracle8, image attributes reside in the same table as the image BLOB. Alternatively, an application designer could define a composite ODT that contains one of the cartridge data types along with other attributes. In subsequent releases, other schemes for handling additional attributes of image data, such as inheritance, may be available.

In collaboration with Virage, a third-party partner, Oracle offers a content-based image retrieval data cartridge based on the Virage VIR engine. This cartridge

allows you to compare images based on the following set of image attributes: global color, local color, texture, and structure.

1.2.3 Accessing Images

Applications access and manipulate images using SQL or PL/SQL through the object relational image types ORDimgB and ORDimgF. The object syntax for accessing attributes within a complex object such as an image is the dot notation:

```
variable.data_attribute
```

The syntax for invoking methods of a complex object such as an image is also the dot notation:

```
variable.function(parameter1, parameter2, ...)
```

A special case exists when using the SELECT, UPDATE, and DELETE statements. You must use a table alias, as follows:

```
SELECT table_alias.column_name.data_attribute FROM table_name table_alias
```

See the *Oracle8 Concepts* manual for information on this and other SQL syntax.

1.3 Image Methods and Operations

Image Cartridge provides methods for extracting image properties, performing image manipulations including format conversion, compression, and data manipulation operations, and copying images. This section presents a conceptual overview of these features. See Chapter 3, “Image Object Types and Procedures” for the syntax of these methods.

1.3.1 Image Property Extraction

The setProperties( ) method is used to extract important properties from image data including file format (such as TIFF, or BMP), content format (monochrome, 8-bit gray scale), height and width in pixels, compression format (JPEG, LZW), and total size in bytes.

This method should be used any time an image is stored in the database, so that the properties are always up to date. For example, the storage size for an image may be machine dependent. Therefore, exporting and then importing an image between systems may require a call to the setProperties( ) method to determine the current properties.

1.3.2 Image Manipulation

The process( ) and processCopy( ) methods are used for image format conversion, compression, and basic manipulation functions including scaling and cropping.

The image may be compressed using an algorithm from the set of supported compression schemes. For example, image data in the TIFF format may be compressed using Packbits, Huffman, JPEG, LZW, or one of the other supported compression schemes. Appendix A, "File and Compression Formats" lists the supported image formats and related compression schemes.

The output of any image manipulation function must be directed to a BLOB. In-place image manipulations of BFILEs is not supported in this release.

1.3.3 Image Copy

The copyContent( ) method is used to copy an image to a new BLOB. The method performs copies to BLOBs using the image content as the source, and a supplied BLOB as the destination. The supplied BLOB must be selected for update.

After copying the image, use the setProperties( ) method to record the attributes of the new image.

2

Using Image Object Types

This chapter provides examples for the common uses of the Image BLOB and Image BFILE types, including:

- adding types to a new or existing table
- inserting a row using BLOB or BFILE images
- populating a row using BLOB or BFILE images
- querying a row
- copying an image from BFILE to BLOB
- copying an image from BLOB to BLOB
- converting an image's format either temporarily or permanently
- extending the cartridge with new object types
- using the cartridge with object views

Prior to updating a BLOB value, you must lock the row containing the BLOB locator. This is usually done using a SELECT FOR UPDATE statement in SQL and PL/SQL programs, or using an OCI pin or lock function in OCI programs.

2.1 Adding Image Types to an Existing Table

Suppose you have an existing table named 'emp' with the following columns:

```
ename      VARCHAR2(50)
salary     NUMBER
job        VARCHAR2(50)
department INTEGER
```

To add a new column to the 'emp' table called 'photo_id' using the Image BLOB type, issue the following statement:

```
ALTER TABLE emp
ADD (photo_id ORDSYS.ORDIMGB);
```

To add a new column to the 'emp' table called 'large_photo' using the Image BFILE type, issue the following statement:

```
ALTER TABLE emp
ADD (large_photo ORDSYS.ORDIMGF);
```

2.2 Adding Image Types to a New Table

Suppose you are creating a new table called 'emp' with the following columns:

```
ename          VARCHAR2(50)
salary         NUMBER
job            VARCHAR2(50)
department     INTEGER
photo_id       ORDIMGB
large_photo    ORDIMGF
```

The column 'photo_id' would use the Image BLOB type, while the column 'large_photo' would use the Image BFILE type. The following statement would create the table:

```
CREATE TABLE emp (
ename VARCHAR2(50),
salary NUMBER,
job VARCHAR2(50),
department INTEGER,
photo_id ORDSYS.ORDIMGB,
large_photo ORDSYS.ORDIMGF);
```

2.3 Inserting a Row Using BLOB Images

To insert a row into a table that has storage for image content using the Image Cartridge BLOB type (ORDImgB), you must populate the type with an initializer. Note that this is different from NULL.

The following examples describe how to insert rows into the table using the Image BLOB type. Assume you have a table 'emp' with the following columns:

```
ename       VARCHAR2(50)
salary      NUMBER
job         VARCHAR2(50)
```

```
department INTEGER
photo_id   ORDIMGB
```

To insert a row into the table with no data in the 'photo_id' column, issue the following statement:

```
INSERT INTO emp VALUES ('John Doe',24000,'Technical Writer',123,NULL);
```

Attempting to use the Image Cartridge types with a NULL value results in an error. If you are going to use the image type's content attribute, you must populate the content attribute with a value and initialize storage for the content attribute with an empty_blob( ) constructor. To insert a row into the table with empty data in the 'photo_id' column, issue the following statement:

```
INSERT INTO emp VALUES ('John Doe',24000,'Technical Writer',123,
ORDSYS.ORDIMGB(empty_blob(),NULL,NULL,NULL,NULL,NULL,NULL));
```

2.4 Populating a Row Using BLOB Images

The following is an example of populating the row with Image BLOB data:

```
DECLARE
    -- application variables
    Image  ORDSYS.ORDIMGB;
BEGIN
    insert into emp values('John Doe',24000,'Technical Writer',123,
    ORDSYS.ORDIMGB(empty_blob(), NULL,NULL,NULL,NULL,NULL,NULL));
    --select the newly inserted row for update
    SELECT photo_id into Image from emp
    where ename = 'John Doe' for UPDATE;

    BEGIN
    -- populate the data with dbms lob calls or write an OCI
    -- program to fill in the content attribute
    END;

    -- set property attributes for the image data
    Image.setProperties;

    UPDATE emp set photo_id = Image where ename = 'John Doe';

    -- continue processing
    END;
```

An UPDATE statement is required to update the property attributes. If you do not perform the setProperties( ) function and UPDATE statement now, you can still commit and the change to the image will be reflected in the content attribute, but not in the properties. See the *Oracle8 Application Developer's Guide* for more information on BLOBs.

2.5 Inserting a Row Using BFILE Images

To insert a row into a table that has storage for image content using the Image BFILE type (ORDImgF), you must populate the type with an initializer. Note that this is different from NULL.

The following examples describe how to insert rows into the table using the Image BFILE type. Assume you have a table 'emp' with the following columns:

```
ename      VARCHAR2(50)
salary     NUMBER
job        VARCHAR2(50)
department  INTEGER
large_photo ORDIMGF
```

To insert a row into the table with no data in the 'large_photo' column, issue the following statement:

```
INSERT INTO emp VALUES ('John Doe',24000,'Technical Writer',123,NULL);
```

Attempting to use the Image BFILE type with a NULL value results in an error. If you are going to use the Image BFILE type column, you must first populate the column with a value. To populate the value of the Image BFILE type column, you must populate the row with a file constructor.

The following example inserts a row into the table with an image called 'jdoe.gif' from the ORDIMGDIR directory:

```
insert into emp values ('John Doe',24000,'Technical Writer',123,
ORDSYS.ORDIMGF(bfilename('ORDIMGDIR','jdoe.gif'),
NULL,NULL,NULL,NULL,NULL,NULL));
```

Note: In release 8.0.3 of the Server, the content of the Image BFILE type is read-only.

The 'bfilename' argument 'ORDIMGDIR' is a directory referring to a file system directory. The following sequence creates a directory named ORDIMGDIR:

```
connect internal
        -- make a directory referring to a file system directory
create directory ordimgdir as '<myimagedirectory>';
grant read on directory ordimgdir to <user-or-role>;
```

where <myimagedirectory> is the the file system directory, and <user-or-role> is the specific user to grant read access to.

2.6 Populating a Row Using BFILE Images

The following is an example of populating the row with Image BFILE data:

```
DECLARE
    Image  ORDSYS.ORDIMGF;
BEGIN
    insert into emp values('John Doe',24000,'Technical Writer',123,
    ORDSYS.ORDIMGF(bfilename('ORDIMGDIR','jdoe.gif'),
    NULL,NULL,NULL,NULL,NULL,NULL));

    --select the newly inserted row for update
    SELECT large_photo into Image from emp
    where ename = 'John Doe' for UPDATE;

    -- set property attributes for the image data
    Image.setProperties;

    UPDATE emp set large_photo = Image where ename = 'John Doe';

    -- continue processing

 END;
```

2.7 Querying a Row

For the following examples, assume you have this table:

```
create table emp (
ename VARCHAR2(50),
salary NUMBER,
job VARCHAR2(50),
department INTEGER,
photo_id ORDSYS.ORDIMGB,
```

```
large_photo ORDSYS.ORDIMGF);
```

The following is an example of querying the row that has Image BFILE data. You must create a table alias (E in this example) when you refer to a type in a SELECT statement.

```
SELECT ename, E.photo_id.width
 FROM emp E
WHERE ename = 'John Doe' and
      E.photo_id.width > 32 and
      E.photo_id.fileFormat='GIFF';
```

The following is an example of querying the row that has Image LOB data:

```
SELECT ename, E.large_photo.compressionFormat
 FROM emp E
WHERE ename = 'John Doe' and
     E.large_photo.width > 32 and
     E.large_photo.fileFormat='GIFF' and
     E.large_photo.compressionFormat='GIFLZW';
```

2.8 Copying an Image from a BFILE to a BLOB Type

To copy the data from an Image BFILE type to an Image BLOB type, you would use the ORDImgF.copyContent method. For example, the following program copies image data from an Image BFILE type to an Image BLOB type:

```
 DECLARE
    BLOBImage ORDSYS.ORDIMGB;
    BFILEImage ORDSYS.ORDIMGF;
 BEGIN
    SELECT photo_id,large_photo
    INTO BLOBImage,FILEImage
    FROM emp where ename = 'John Doe' for UPDATE;

    -- Copy the BFILE image to the BLOB image
    BFILEImage.copyContent(BLOBImage.content);

    -- Set the BLOB image properties
    BLOBImage.setProperties;

    -- update the row
    UPDATE emp
      SET photo_id = BLOBImage
      where ename = 'John Doe';
```

```
END
```

2.9 Copying an Image from a BLOB to a BLOB Type

To copy the data between two Image BLOB types, use the ORDImgB.copyContent method. For example, the following program copies image data from an Image BLOB type to another Image BLOB type:

```
DECLARE
    Image_1 ORDSYS.ORDIMGB;
    Image_2 ORDSYS.ORDIMGB;
BEGIN
    SELECT photo_id
    INTO Image_1
    FROM emp where ename = 'John Doe';

    SELECT photo_id
    INTO Image_2
    FROM emp where ename = 'Also John Doe' for UPDATE;

    -- copy the data from Image_1 to Image_2
    Image_1.copyContent(Image_2.content);

    -- set the image properties for Image_2
    Image_2.setProperties;

    -- continue processing

    UPDATE emp
      SET photo_id = Image_2
      WHERE ename = 'Also John Doe';

END
```

2.10 Converting an Image's Format

To convert the image data into a different format, use the Process method. For example, the following program converts the image data to the TIFF file format:

```
DECLARE
   Image ORDSYS.ORDIMGB;
BEGIN
   SELECT photo_id
   INTO Image
   FROM emp
```

```
    WHERE ename = 'John Doe' for UPDATE;

    -- convert the image to TIFF in place
    Image.process('fileFormat=TIFF');

END
```

2.11 Copying and Converting in One Step

To make a copy of the image and convert it into one step, use the processCopy method. For example, the following program converts the image data to the TIFF image file format, but leaves the original image intact:

```
DECLARE
    Image_1 ORDSYS.ORDIMGB;
    Image_2 ORDSYS.ORDIMGB;
BEGIN
    SELECT photo_id
    INTO Image_1
    FROM emp
    WHERE ename = 'John Doe' for UPDATE;

    -- convert the image to tiff and store the result in Image_2
    Image_2 := Image_1;
    Image_1.processCopy('fileFormat=TIFF',Image_2.content);

    -- continue processing

END
```

Changes made by these methods can be rolled back. This technique may be useful for a temporary format conversion.

2.12 Extend the Cartridge With a New Type

You can use the ORDImgF and ORDImgB types as the basis for a new type of your own creation. This task requires the Objects Option of Oracle8 Enterprise Edition.

```
create type AnnotatedImage as object
   (  image ordsys.ordimgF,
       description varchar2(2000),

      MEMBER PROCEDURE SetProperties(SELF IN OUT AnnotatedImage),
      MEMBER PROCEDURE CopyContent(dest IN OUT BLOB),
      MEMBER PROCEDURE ProcessCopy(command in VARCHAR2, dest IN OUT BLOB)
    );
```

```
/

create type body AnnotatedImage as
  MEMBER PROCEDURE SetProperties(SELF IN OUT AnnotatedImage) IS
  BEGIN
    SELF.image.setProperties;
    SELF.description :=
 'This is an example of using Image Cartridge as a subtype';
  END SetProperties;

  MEMBER PROCEDURE CopyContent(dest IN OUT BLOB) IS
  BEGIN
    SELF.image.copyContent(dest);
  END CopyContent;

  MEMBER PROCEDURE ProcessCopy(command in VARCHAR2, dest IN OUT BLOB) IS
  BEGIN
    SELF.image.processCopy(command,dest);
  END ProcessCopy;
END;
/
```

After creating the new type, you can use it as you would any other type. For example:

```
create or replace directory TEST_DIR as 'C:\TESTS';

create table my_example (id number,an_image AnnotatedImage);

insert into my_example values ( 1,
    AnnotatedImage(
      ordsys.ordimgf(
        bfilename('TEST_DIR','test1.jpg'),
        NULL,NULL,NULL,NULL,NULL,NULL,NULL),
      NULL)
    );
commit;

declare
  myimage AnnotatedImage;
begin
  select an_image into myimage from my_example;

  myimage.setProperties;
```

```
  dbms_output.put_line('This image has a description of ');
  dbms_output.put_line( myimage.description);

  update my_example set an_image=myimage;
end;
```

2.13 Use Image Types With Object Views

Just as a view is a virtual table, an object view is a virtual object table.

Oracle provides object views as an extension of the basic relational view mechanism. By using object views, you can create virtual object tables from data-- of either built-in or user-defined types -- stored in the columns of relational or object tables in the database.

Object views provide the ability to offer specialized or restricted access to the data and objects in a database. For example, you might use an object view to provide a version of an employee object table that doesn't have attributes containing sensitive-data and doesn't have a deletion method. Object views also allow you to try object-oriented programming without permanently converting your tables. Using object views, you can convert data gradually and transparently from relational tables to object-relational tables.

Consider the following non-object image table:

```
create table flat (
   id            number,
   content       bfile,
   height        number,
   width         number,
   contentLength number,
   fileFormat    varchar2(64),
   contentFormat varchar2(64),
   compressionFormat varchar2(64)
                  );
```

You can create an object view on the flat table as follows:

```
create or replace view object_images_v as
  select
      id,
      ordsys.ORDImgF(
         T.content,
         T.height,
```

```
        T.width,
        T.contentLength,
        T.compressionFormat
                    ) IMAGE
from flat T;
```

Object views provide the flexibility of looking at the same relational or object data in more than one way. Thus you can use different in-memory object representations for different applications without changing the way you store the data in the database. See the *Oracle8 Concepts* manual for more information on defining, using, and updating object views.

3

Image Object Types and Procedures

Oracle8 Image Cartridge consists of two object data types:

- ORDImgB supports images stored in an Oracle8 binary large object (BLOB)
- ORDImgF supports images stored in an Oracle8 external binary file (BFILE)

The cartridge includes four user-visible procedures:

- copyContent: creates a copy of an image in another BLOB (available only for BLOBs, and not BFILEs)
- process: performs in-place image processing on a BLOB
- processCopy: performs image processing while copying an image to another BLOB
- setProperties: fills in the attribute fields of an image (ORDImgB or ORDImgF data type)

When you are storing or copying images, you must first create an empty BLOB in the table. The examples in this chapter assume that the following table, ordimgtab, has been created to store three images. Three empty rows have been created as follows:

```
create table ordimgtab(col1 number, col2 ORDSYS.ORDImgB);
grant select, update on ordimgtab to ordsys;
insert into ordimgtab values
  (1, ORDSYS.ORDImgB(empty_blob(), NULL, NULL, NULL, NULL, NULL, NULL));
insert into ordimgtab values
  (2, ORDSYS.ORDImgB(empty_blob(), NULL, NULL, NULL, NULL, NULL, NULL));
insert into ordimgtab values
  (3, ORDSYS.ORDImgB(empty_blob(), NULL, NULL, NULL, NULL, NULL, NULL));
commit;
```

ORDImgB Object Data Type

The ORDImgB object data type (ODT) is used for basic storage and retrieval of image data within an Oracle database. This object data type is defined as follows:

```
CREATE TYPE ORDImgB AS OBJECT
(
-- TYPE ATTRIBUTES
content             BLOB,
height              INTEGER,
width               INTEGER,
contentLength       INTEGER,
fileFormat          VARCHAR2(64),
contentFormat       VARCHAR2(64),
compressionFormat   VARCHAR2(64),
--- METHOD DECLARATION
  MEMBER PROCEDURE copyContent(dest IN OUT ORDImgB),
  MEMBER PROCEDURE setProperties(SELF IN OUT ORDImgB),
  MEMBER PROCEDURE process    (SELF    IN OUT ORDImgB,
                               command  IN     VARCHAR2)
  MEMBER PROCEDURE processCopy(command  IN     VARCHAR2,
                               dest     IN OUT BLOB)
);
```

Where:

- content: Stored image
- height: Height of the image in pixels
- width: Width of image in pixels
- contentLength: Size of the *on-disk* image file in bytes
- fileFormat: File type of image (such as, TIFF, JFIF)
- contentFormat: Type of image (such as, monochrome, 8-bit gray scale)
- compressionFormat: Compression type of image

In PL/SQL data is moved with the DBMS LOB package. From the client, data is moved via OCI LOB calls. The ORDimgB ODT does not supply piece-wise routines for moving data.

ORDImgF Object Data Type

The ORDImgF object data type is used for retrieval of image data stored in external files. BFILE images are assumed to be read-only and this is reflected in the member procedures defined on the object data type.

```
CREATE TYPE ORDImgF AS OBJECT
(
-- TYPE ATTRIBUTES
content             BFILE,
height              INTEGER,
width               INTEGER,
contentLength       INTEGER,
fileFormat          VARCHAR2(64),
contentFormat       VARCHAR2(64),
compressionFormat   VARCHAR2(64),

-- METHOD DECLARATION
MEMBER PROCEDURE copyContent(dest IN OUT ORDImgB),
MEMBER PROCEDURE setProperties(SELF IN OUT ORDImgF),
MEMBER PROCEDURE processCopy(command IN VARCHAR2,
                             dest    IN OUT BLOB)
);
```

Where:

- content: Stored image
- height: Height of the image in pixels
- width: Width of image in pixels
- contentLength: Size of the *on-disk* image file in bytes
- fileFormat: File type of image (such as, TIFF, JFIF)
- contentFormat: Type of image (such as, monochrome, 8-bit gray scale)
- compressionFormat: Compression type of image

copyContent Procedure

Format

copyContent (dest IN OUT BLOB);

Description

Copy an image without changing it.

Parameters

dest
The destination of the new image.

Usage

Copies the image data into the supplied BLOB.

Example

Create a copy of the image in type image1 into a BLOB called myblob:

```
image1.copyContent(myblob);
```

process Procedure

Format

process (command IN VARCHAR2);

Description

Perform one or more image processing techniques on a BLOB, writing the image back on itself.

Parameters

command
A list of image processing changes to make for the image.

Usage

Change one or more of the image attributes shown in the following table:

Table 3–1 Image Processing Parameters

Parameter Name	Usage	Values
fileFormat	file format of the image	GIFF, TIFF, PCXF, PICT, RASF, CALS, BMPF, TGAF, JFIF
contentFormat	imagetype / pixel / data format.	MONOCHROME, RAW, 4BITGRAYSCALE, 4BITGREYSCALE, 8 BITGRAYSCALE, 8 BITGREYSCALE, 1BITLUT, 2BITLUT, 4 BITLUT, 8BITLUT, 16BITRGB, 24BITRGB, 32BITRGB, 24BITPLANAR
scale	scale factor (for example, 0.5 or 2.0)	<FLOAT> positive
xscale	X-axis scale factor (default is 1)	<FLOAT> positive
yscale	Y-axis scale factor (default is 1)	<FLOAT> positive

Table 3–1 Image Processing Parameters

Parameter Name	Usage	Values
compressionQuality	compression quality	MAXCOMPRATIO, MAXINTEGRIT, LOWCOMP, MEDCOMP, HIGHCOMP
compressionFormat	compression type/format	JPEG, SUNRLE, BMPRLE, TARGARLE, LZW, LZWHDIFF, FAX3, FAX4, HUFFMAN3, Packbits, GIFLZW
cut	Window to cut or crop (origin.x origin.y width height)	(Integer Integer Integer Integer) limit < 65535

Note: When specifying parameter values that include floating-point numbers, you must use double quotation marks (" ") around the value. If you do not, this may result in incorrect values being passed and you will get incorrect results.

Examples

Change the file format of image1 to GIF:

```
image1.process('fileFormat=GIFF');
```

Change image1 to use lower quality JPEG compression and double the length of the image along the X-axis:

```
image1.process('compressionFormat=JPEG, compressionQuality=LOWCOMP,
xscale="2.0"');
```

processCopy Procedure

Format

processCopy (command IN VARCHAR2,

dest IN OUT BLOB);

Description

Copy an image BLOB to another BLOB.

Function Parameters

command
A list of image processing changes to make for the image in the new copy.

dest
The destination of the new image.

Usage

See Table 3–1, “Image Processing Parameters”.

Example

Copy an image, changing the file format, compression format, and data format in the destination image:

```
create or replace procedure is
  imgB1     ORDSYS.ORDImgB;
  imgB4     ORDSYS.ORDImgB;
  mycommand   VARCHAR2(400);
begin
  select col2 into imgB1 from ordimgtab  where col1 = 1;
  select col2 into imgB4 from ordimgtab  where col1 = 4 for update;
  command:= 'fileFormat=tiff compressionFormat = packbits
  contentFormat = 8bitlut';
  imgB1.processcopy(mycommand,imgB4.content);
  imgB4.setproperties;
end;
```

setProperties Procedure

Format

setProperties();

Description

Write the characteristics of an image (BLOB or BFILE) into the appropriate attribute fields.

Parameters

None

Usage

After you have copied or stored an image, call this procedure to set the current characteristics of the new content.

This procedure sets the following information about an image:

- height in pixels
- width in pixels
- data size of the on-disk image in bytes
- file type (TIFF, JFIF, and so forth)
- image type (monochrome, 8-bit gray scale, and so forth)
- compression type (JPEG, LZW, and so forth)

Example

Select the type, and then set the attributes using the setProperties procedure.

```
imgB1 ORDSYS.imgB;
.
.
.
select col2 into imgB1 from ordimgtab where col1 = 1 for update;
imgB1.setProperties;
dbms_output.put_line('image width = '|| imgB1.width );
dbms_output.put_line('image height = '|| imgB1.height );
dbms_output.put_line('image size = '|| imgB1.contentLength );
```

```
dbms_output.put_line('image file type = '|| imgB1.fileFormat );
dbms_output.put_line('image type = '|| imgB1.contentFormat );
dbms_output.put_line('image compression = '|| imgB1.compressionFormat );
```

Example output:

```
image width = 360
image height = 490
image size = 59650
image file type = JFIF
image type = 24BITRGB
image compression = JPEG
```

A

File and Compression Formats

A.1 Supported File and Compression Formats

The following tables describe the file and compression formats supported by Image Cartridge.

To use these tables, find the data format you are interested in, and then determine the supported formats. For example, Table A–1 shows that Image Cartridge supports BMP format in monochrome, for read and write access, and in 32-bit RGB for read access.

Table A–1 BMP Data Format

Format	Pixel Format	Support
BMP	Monochrome	Read / Write
	4-bit LUT	Read
File Format: 'BMPF'	8-bit LUT	Read / Write
File Ext: .bmp	16-bit RGB	Read
	24-bit RGB	Read / Write
	32-bit RGB	Read
	Compression Format	**Support**
Choose one of these compression formats:	uncompressed	Read / Write
	BMPRLE (for 8-bit LUT)	Read / Write

	Data Description	Support
Choose one or more of these content formats:	Inverse DIB	Read
	OS/2 format	Read
	Byte Order	**Support**
	NA	NA

Table A–2 CALS Raster Data Format

Format	Pixel Format	Support
CALS Raster File Format: 'CALS' File Ext: .cal	Monochrome	Read/Write
	Compression Format	**Support**
	FAX4 (CCITT G4)	Read/Write
	Data Description	**Support**
	NA	NA
	Byte Order	**Support**
	NA	NA

Table A–3 GIF Data Format

Format	Pixel Format	Support
GIF File Format: 'GIFF' File Ext: .gif	Monochrome 8-bit LUT	Read Read / Write
	Compression Format	**Support**
	GIFLZW (LZW)	Read / Write
	Data Description	**Support**
	NA	NA
	Byte Order	**Support**
	NA	NA

Table A–4 JFIF Data Format

Format	Pixel Format	Support
JFIF File Format: 'JFIF' File Ext: .jpg	8-bit grayscale 24-bit RGB	Read / Write Read / Write
	Compression	**Support**
	JPEG	Read / Write
	Data Description	**Support**
	NA	NA
	Byte Order	**Support**
	NA	NA

Table A–5 PCX Data Format

Format	Pixel Format	Support
PCX v 5	Monochrome	Read
	2-bit LUT	Read
File Format: 'PCXF'	4-bit LUT	Read
File Ext: .pcx	8-bit LUT	Read
	1-bit RGB	Read
	2-bit RGB	Read
	4-bit RGB	Read
	8-bit RGB	Read
	24-bit RGB	Read
	Compression Format	**Support**
	RLE	Read
	Data Description	**Support**
	NA	NA
	Byte Order	**Support**
	NA	NA

Table A–6 PICT Data Format

Format	Pixel Format	Support
PICT v. 1 & 2	Monochrome	Read / Write
	2-bit LUT	Read
File Format: 'PICT'	4-bit LUT	Read
File Ext: .pct	8-bit LUT	Read / Write
	16-bit RGB	Read
	24-bit RGB	Read / Write

	Compression Format	Support
Choose one of these compression formats:	Packbits	Read / Write
	JPEG (8-bit gray & RGB)	Read / Write
	Data Description	**Support**
Choose one or more of these content formats:	vector / object graphics	Not supported
	Byte Order	**Support**
	NA	NA

Table A–7 Sun Raster Data Format

Format	Pixel Format	Support
Sun Raster	Monochrome	Read / Write
	8-bit grayscale	Read / Write
File Format: 'RASF'	8-bit LUT	Read / Write
File Ext: .ras	24-bit RGB	Read / Write
	Compression Format	**Support**
Choose one of these compression formats:	(uncompressed)	Read / Write
	SUNRLE (RLE)	Read / Write
	Data Description	**Support**
	NA	NA
	Byte Order	**Support**
	NA	NA

Table A–8 Targa Data Format

Format	Pixel Format	Support
Targa	8-bit grayscale	Read / Write
	8-bit LUT	Read / Write
File Format:'TGAF'	16-bit RGB	Read
File Ext: .tga	24-bit RGB	Read / Write
	32-bit RGB	Read
	Compression Format	**Support**
Choose one of these compression formats:	(uncompressed)	Read / Write
	TARGARLE (RLE)	Read / Write
	Data Description	**Support**
	NA	NA
	Byte Order	**Support**
	NA	NA

Table A–9 TIFF Data Format

Format	Pixel Format	Support
TIFF v.4/5/6	Monochrome	Read / Write
	8-bit grayscale	Read / Write
File Format: 'TIFF'	4 bit LUT	Read
File Ext: .tif	8-bit LUT	Read / Write
	24-bit RGB	Read / Write
	Compression Format	**Support**
Choose one of these compression formats:	(uncompressed)	Read / Write
	Packbits	Read / Write
	Huffman	Read / Write
	FAX3 (CCITT G3)	Read / Write
	FAX4 (CCITT G4)	Read / Write
	LZW	Read / Write
	LZWHDIFF	Read / Write
	JPEG (8-bit gray & RGB)	Read / Write
	Data Description	**Support**
Choose one or more of these content formats:	Planar data	Not supported
	Tiled data	Not supported
	Photometric interpretation	Read / Write
	MSB / LSB	Read / Write
	Byte Order	**Support**
Choose one of these orders:	Intel byte order	Read
	Motorola byte order	Read / Write

B

Installation and Demo

Oracle8 Image Cartridge is installed under the database user ORDSYS. This user is created during installation and is subject to change in future beta or production releases.

If you created database objects during the Image Cartridge installation, the password for the ORDSYS user may have been written to the log file. This will happen if you attempt to re-install the data cartridge database objects more than one time. Please check your installation log files.

B.1 Installation Procedure

This section describes the generic steps required to install Image Cartridge. See your platform-specific documentation for details.

B.1.1 Pre-Installation Decisions

During the installation process you are prompted to set the password for the ORDSYS user. This user ID is the standard Oracle database account with special privileges for data cartridges. You must decide on a password for the ORDSYS user.

B.1.2 Pre-Installation Steps

Perform the following tasks prior to installing Oracle data cartridges. For instructions, please see the Oracle8 installation and configuration guide for your operating system.

1. Install the Oracle8 RDBMS, including the PL/SQL option.
2. Create the database.
3. Start up the database.

B.1.3 Installation Steps

Perform the following mandatory installation steps. If you are using the Oracle Installer with the Create New Database Objects option, then steps 2 through 6 will be done for you automatically.

1. Install the non-database components for Image Cartridge using the Oracle Installer.
2. Create the ORDSYS account.

   Create the ORDSYS account. Use the password you gave in Section B.2.1.

   ```
   SVRMGRL> create user ORDSYS identified by <ORDSYS password>;
   ```

3. Grant privileges to the ORDSYS account.

   Grant the following privileges to the ORDSYS account:

   ```
   SVRMGRL> grant connect,resource,create library to ORDSYS;
   ```

4. Create the image library where <ORACLE_HOME> is the ORACLE_HOME directory. .

   ```
   SVRMGRL> connect ORDSYS/<ORDSYS password>
   SVRMGRL> create or replace library ORDImgLibS as <ORACLE_HOME>/lib/
   libordimg.so';
   ```

   The exact name of this library is operating system dependent. See the installation and configuration guide for your operating system for the exact file name.

5. Create the cartridge where <ORACLE_HOME> is the ORACLE_HOME directory.

   ```
   SVRMGRL> connect ORDSYS/<ORDSYS password>
   SVRMGRL> @<ORACLE_HOME>/ord/img/admin/ordispec.sql
   SVRMGRL> @<ORACLE_HOME>/ord/img/admin/ordibody.plb
   ```

6. Grant privileges to others on the Image Cartridge data types.

   ```
   % svrmgrl
   SVRMGRL> connect ORDSYS/<ORDSYS password>
   SVRMGRL> grant execute on ORDSYS.ORDImgB to PUBLIC;
   SVRMGRL> grant execute on ORDSYS.ORDImgF to PUBLIC;
   ```

B.2 Demo Program

Once you have installed Image Cartridge, you may choose to install the Image Cartridge demonstration program. The demo can be used as a test to confirm successful installation.

This section contains the steps required to install the Image Cartridge demo.

The image demo files are located in <ORACLE_HOME>/ord/img/demo, where <ORACLE_HOME> is the ORACLE_HOME directory.

B.2.1 Demo Installation Steps

1. The image cartridge demo uses the SCOTT/TIGER database user. If this user does not exist, you must create it:

```
% svrmgrl
SVRMGRL> connect internal;
SVRMGRL> create user SCOTT identified by tiger;
SVRMGRL> grant connect,resource to SCOTT;
```

2. Create the image demo directory where <ORACLE_HOME> is the ORACLE_HOME directory.

```
% svrmgrl
SVRMGRL> connect internal;
SVRMGRL> create or replace directory imgdemodir as '<ORACLE_HOME>/ord/img
demo';
```

3. Grant privileges on the directory to PUBLIC.

```
SVRMGRL> grant read on directory imgdemodir to public with grant option;
```

4. If needed, make the imgdemo program.

```
% cd <ORACLE_HOME>/ord/img/demo
% make -f demo_ordimg.mk imgdemo
```

   The make operation is operating system specific. See the installation and configuration guide for your operating system for details on how to make this program on your operating system.

B.2.2 Running the Demo

The file imgdemo is a sample program that shows how Image Cartridge can be used from within a program. The demo is written in C and uses OCI, Oracle Call-Interface™, to access the database and exercise the cartridge.

The program operates on imgdemo.dat, which is a bitmap (BMP) image in the local directory. Optionally, you can supply an image file name on the command line, provided that the file resides in the same directory as the demo. In either case, once the image has been manipulated by the cartridge, the resulting image is written to the file imgdemo.out and can then be viewed with common rendering tools that you supply.

When the demo is run, it drops and re-creates a table named IMGDEMOTAB in the SCOTT/TIGER account of the default database. This table is used to hold the demo data. Once the table is created, a reference to the image file is inserted into the table. The data is then loaded into the table using the copyContent( ) method of the ORDImgF cartridge type.

The image properties are extracted within the database using the setProperties( ) method. An UPDATE command is issued after the setProperties( ) invocation. This is required because the setProperties( ) invocation has only updated a local copy of the type attributes.

Next, the Image Cartridge process( ) method is used to cut and scale the image within the database. This is followed by an update that commits the change. The program cut a portion of the image 100 pixels wide by 100 pixels high starting from pixel location (100,100). This subimage is scaled to twice its original size and the resulting image is written out to the file system in a file named imgdemo.out.

Upon completion, the demo program leaves the imgdemo.out file in the current directory. It also leaves the table IMGDEMOTAB in the SCOTT/TIGER account of the database.

Example B–1

Execute the demo by typing imgdemo on the command line. Optionally, a different image can be used in the demo by first copying the file to the directory in which the demo resides and then specifying its file name on the command line as an argument to imgdemo.

Use the following command:

```
$ imgdemo <optional-image-filename>
```

The demo displays a number of messages describing its progress, along with any errors encountered in the event that something was not set up correctly. Expect to see the following messages:

```
Dropping table IMGDEMOTAB...
Creating and populating table IMGDEMOTAB...
Loading data into the cartridge...
Modifying image characteristics...
Writing image to file imddemo.out...
Disconnecting from database...
Logged off and detached from server.
Demo completed successfully.
```

If the program encounters any errors, it is likely that either the Image Cartridge software has not been installed correctly or the database has not been started. If the program completes successfully, the original image and the resultant image, which has undergone the cutting and scaling described earlier, can be viewed with common image rendering tools.

B.3 Image Demo Code Example

The following code example represents the demonstration program shipped with this cartridge. This example was designed to work on UNIX systems. A different version may be available on your platform.

```
#ifdef RCSID
static char *RCSid =
   "$Header: imgdemo.c 18-apr-97.10:26:57 svivian Exp $ ";
#endif /* RCSID */

/* Copyright (c) Oracle Corporation 1997. All Rights Reserved. */
/*
   NAME
     imgdemo.c - out-of-the-box demo using the image data cartridge

   DESCRIPTION
      This program demonstrates how to use the image data cartidge to
      load an image into the database, alter its attributes, and then
      write the data to a file for viewing.  The demo will use an
      image data file provided with the cartridge named imgdemo.dat
      as its input so no parameter is required.

      The process method will be used to scale and crop the original
      image so expect the resultant image in imgdemo.out to reflect
```

```
   this change.

   Sample usage:    imgdemo <optional-image-filename>

   imgdemo.dat    --  image to be manipulated
   imgdemo.out    --  resultant image filename

   Optionally, a user provided image file can be specified on
   the command line as shown above, provided that it resides
   in the IMGDEMODIR directory (see assumptions below).

ENVIRONMENT:

   The following assumptions are made:

    1) Image data cartridge has been installed and PUBLIC
       has EXECUTE privilege on it.

    2) Install script has been run and thus created IMGDEMODIR
       directory and granted public read access in order that
       the image datafile can be read into the database.

    3) imgdemo.dat is a valid image file that resides in the
       IMGDEMODIR directory and the user has read/write access
       to the directory.

    4) User SCOTT has the default password

PUBLIC FUNCTION(S)

PRIVATE FUNCTION(S)

RETURNS

NOTES
   After demo has been executed, use a rendering tool such as xv
   to view both the original and the modified images.  The table
   IMGDEMOTAB is left in the SCOTT account in the default database
   for the user's viewing as well.  The IMGDEMODIR directory will
   likewise be left in the system account.

MODIFIED   (MM/DD/YY)
svivian     05/19/97 - call setProperties after modifying image
svivian     05/15/97 - NT portability - open file for binary write
svivian     04/24/97 - change select to use table alias
```

```
   svivian     04/18/97 - Creation

*/

 #include <stdio.h>

#ifndef OCI_ORACLE
#include <oci.h>
#endif

/* local routines */
static sb4 init_handles();
static sb4 attach_server();
static sb4 log_on();
static void logout();
static sb4 create_table();
static void drop_table();
static sb4 fetch_blob();
static sb4 load_cart();
static sb4 mod_img();
static sb4 blob_to_file();
static void report_error();

#define TRUE       1
#define FALSE      0

#define MAXBUFLEN  16384
#define STMTLEN    512
#define FNAMELEN   80

static OCIEnv        *envhp;
static OCIServer     *srvhp;
static OCISvcCtx     *svchp;
static OCIError      *errhp;
static OCISession    *authp;
static OCIStmt       *stmthp;
static OCILobLocator *blob, *bfile;
static OCIDefine     *defnp1, *defnp2;
static OCIBind       *bndhp;

static char *fname = "imgdemo.out";
static text *user = (text *)"SCOTT";
static text *upwd = (text *)"TIGER";
```

```
int main(int argc, char *argv[])
{
  char iname[FNAMELEN];
  if (argc == 2)
  {
     strcpy(iname, argv[1]);
  }
  else
  {
     strcpy(iname, "imgdemo.dat");
  }

  if (init_handles())
  {
    (void) fprintf(stdout,"FAILED: init_handles()\n");
    return OCI_ERROR;
  }

  if (attach_server())
  {
    (void)fprintf (stdout, "FAILED: attach_server()\n");
    logout();
    return OCI_ERROR;
  }

  if (log_on(user,upwd))
  {
    (void) fprintf(stdout,"FAILED: log_on()\n");
    logout();
    return OCI_ERROR;
  }

  if (create_table(iname))
  {
    (void) fprintf(stdout,"FAILED: create_table()\n");
    logout();
    return OCI_ERROR;
  }

  (void)fprintf (stdout, "\nLoading data into cartridge...\n");
  if (load_cart())
  {
    (void) fprintf(stdout,"FAILED: load_cart()\n");
    logout();
    return OCI_ERROR;
```

```
  }

  (void)fprintf (stdout, "\nModifying image characteristics...\n");
  if (mod_img())
  {
    (void)fprintf (stdout, "\nFAILED: mod_img()\n");
    logout();
    return OCI_ERROR;
  }

  (void)fprintf (stdout, "\nWriting image to file %s...\n",fname);
  if (blob_to_file())
  {
    (void)fprintf (stdout, "\nFAILED: blob_to_file()\n");
    logout();
    return OCI_ERROR;
  }

  (void)fprintf (stdout, "\nDisconnecting from database...\n");
  logout();

  (void)fprintf (stdout, "\nDemo completed successfully.\n");
  return OCI_SUCCESS;

}  /* end main */


/*
 * ---------------------------------------------------------------
 * init_handles - initialize environment, allocate handles, etc.
 * ---------------------------------------------------------------
 */

sb4 init_handles()
{
  sword status;

  if (OCIInitialize((ub4) OCI_DEFAULT,
                    (dvoid *)0,
                    (dvoid * (*)(dvoid *, size_t)) 0,
                    (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                    (void (*)(dvoid *, dvoid *)) 0 ))
  {
    (void) fprintf(stdout,"FAILED: OCIInitialize()\n");
    return OCI_ERROR;
```

```
}

/* initialize environment handle */
if (OCIEnvInit((OCIEnv **) &envhp,
               (ub4) OCI_DEFAULT,
               (size_t) 0,
               (dvoid **) 0 ))
{
  (void) fprintf(stdout,"FAILED: OCIEnvInit()\n");
  return OCI_ERROR;
}

if (OCIHandleAlloc((dvoid *) envhp,
                   (dvoid **) &svchp,
                   (ub4) OCI_HTYPE_SVCCTX,
                   (size_t) 0,
                   (dvoid **) 0))
{
  (void) fprintf(stdout,"FAILED: OCIHandleAlloc()\n");
  return OCI_ERROR;
}

if (OCIHandleAlloc((dvoid *) envhp,
                   (dvoid **) &errhp,
                   (ub4) OCI_HTYPE_ERROR,
                   (size_t) 0,
                   (dvoid **) 0))
{
  (void) fprintf(stdout,"FAILED: OCIHandleAlloc()\n");
  return OCI_ERROR;
}

if (OCIHandleAlloc((dvoid *) envhp,
                   (dvoid **) &stmthp,
                   (ub4) OCI_HTYPE_STMT,
                   (size_t) 0,
                   (dvoid **) 0))
{
  (void) fprintf(stdout,"FAILED: OCIHandleAlloc()\n");
  return OCI_ERROR;
}

if (OCIHandleAlloc((dvoid *) envhp,
                   (dvoid **) &srvhp,
                   (ub4) OCI_HTYPE_SERVER,
```

```
                    (size_t) 0,
                    (dvoid **) 0))
{
  (void) fprintf(stdout,"FAILED: OCIHandleAlloc()\n");
  return OCI_ERROR;
}

if (OCIHandleAlloc((dvoid *) envhp,
                    (dvoid **) &authp,
                    (ub4) OCI_HTYPE_SESSION,
                    (size_t) 0,
                    (dvoid **) 0))
{
  (void) fprintf(stdout,"FAILED: OCIHandleAlloc()\n");
  return OCI_ERROR;
}

/* allocate the lob locator variables */
if (OCIDescriptorAlloc((dvoid *) envhp,
                       (dvoid **) &blob,
                       (ub4)OCI_DTYPE_LOB,
                       (size_t) 0,
                       (dvoid **) 0))
{
  (void) fprintf(stdout,"FAILED: OCIDescriptorAlloc(blob)\n");
  return OCI_ERROR;
}

/* allocate the lob locator variables - will change to OCI_DTYPE_FILE */
if (OCIDescriptorAlloc((dvoid *) envhp,
                       (dvoid **) &bfile,
                       (ub4)OCI_DTYPE_LOB,
                       (size_t) 0,
                       (dvoid **) 0))
{
  (void) fprintf(stdout,"FAILED: OCIDescriptorAlloc(bfile)\n");
  return OCI_ERROR;
}

/* allocate the define handles */
if (OCIHandleAlloc((dvoid *) stmthp,
                    (dvoid **) &defnp1,
                    (ub4) OCI_HTYPE_DEFINE,
                    (size_t) 0,
                    (dvoid **) 0))
```

```
  {
    (void) fprintf(stdout,"FAILED: OCIHandleAlloc()\n");
    return OCI_ERROR;
  }

  if (OCIHandleAlloc((dvoid *) stmthp,
                     (dvoid **) &defnp2,
                     (ub4) OCI_HTYPE_DEFINE,
                     (size_t) 0,
                     (dvoid **) 0))
  {
    (void) fprintf(stdout,"FAILED: OCIHandleAlloc()\n");
    return OCI_ERROR;
  }

  /* allocate the bind handle */
  if (OCIHandleAlloc((dvoid *) stmthp,
                     (dvoid **) &bndhp,
                     (ub4) OCI_HTYPE_BIND,
                     (size_t) 0,
                     (dvoid **) 0))
  {
    (void) fprintf(stdout,"FAILED: OCIHandleAlloc()\n");
    return OCI_ERROR;
  }

  return OCI_SUCCESS;

}  /* end init_handles */


/*
 * ---------------------------------------------------------------------
 * attach_server - attach to default server
 * ---------------------------------------------------------------------
 */
sb4 attach_server()
{
  /* attach to the server - use default host */
  if (OCIServerAttach(srvhp,
                      errhp,
                      (text *) NULL,
                      0,
                      (ub4) OCI_DEFAULT))
  {
```

```
    (void) fprintf(stdout,"FAILED: OCIServerAttach()\n");
    return OCI_ERROR;
  }

  /* set the server attribute in the service context */
  if (OCIAttrSet((dvoid *) svchp,
                 (ub4) OCI_HTYPE_SVCCTX,
                 (dvoid *) srvhp,
                 (ub4) 0,
                 (ub4) OCI_ATTR_SERVER,
                 errhp))
  {
    (void) fprintf(stdout,"FAILED: OCIAttrSet()\n");
    return OCI_ERROR;
  }

  return (OCI_SUCCESS);

}  /* end attach_server */


/*
 * ---------------------------------------------------------------------
 * log_on - log on to server
 * ---------------------------------------------------------------------
 */
sb4 log_on(
  text *uid,
  text *pwd
)
{
  if (OCIAttrSet((dvoid *) authp,
                 (ub4) OCI_HTYPE_SESSION,
                 (dvoid *) uid,
                 (ub4) strlen((char *)uid),
                 (ub4) OCI_ATTR_USERNAME,
                 errhp))
  {
    (void) fprintf(stdout,"FAILED: OCIAttrSet()\n");
    return OCI_ERROR;
  }

  if (OCIAttrSet((dvoid *) authp,
                 (ub4) OCI_HTYPE_SESSION,
                 (dvoid *) pwd,
```

```
                (ub4) strlen((char *)pwd),
                (ub4) OCI_ATTR_PASSWORD,
                errhp))
  {
    (void) fprintf(stdout,"FAILED: OCIAttrSet()\n");
    return OCI_ERROR;
  }

  /* log on */
  if (OCISessionBegin(svchp,
                      errhp,
                      authp,
                      (ub4) OCI_CRED_RDBMS,
                      (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: OCISessionBegin()\n");
    return OCI_ERROR;
  }

  /* set the session attribute in the service context */
  if (OCIAttrSet((dvoid *) svchp,
                (ub4) OCI_HTYPE_SVCCTX,
                (dvoid *) authp,
                (ub4) 0,
                (ub4) OCI_ATTR_SESSION, errhp))
  {
    (void) fprintf(stdout,"FAILED: OCIAttrSet()\n");
    return OCI_ERROR;
  }

  return OCI_SUCCESS;

}  /* end log_on */


/*
 * ---------------------------------------------------------------
 * create_table - Create table IMGDEMOTAB and insert one row.
 * ---------------------------------------------------------------
 */

sb4 create_table(char *iname)
{
  text *crtstmt = (text *)
"CREATE TABLE IMGDEMOTAB (C1 INT, C2 ORDSYS.ORDImgF, C3 ORDSYS.ORDImgB)";
```

```
  text insstmt[STMTLEN];

 sprintf ((char*)insstmt,"INSERT INTO IMGDEMOTAB VALUES
(1,ORDSYS.ORDImgF(bfilename('IMGDEMODIR','%s')
NULL,NULL,NULL,NULL,NULL,NULL),ORDSYS.ORDImgB(empty_blob(),NULL,NULL,
NULL,NULL,NULL,NULL))",iname);

  /*
   * drop table first if it exists and then re-create.
   */
  drop_table();

  /*
   * create table
   */
  if (OCIStmtPrepare(stmthp,
                     errhp,
                     crtstmt,
                     (ub4) strlen((char *) crtstmt),
                     (ub4) OCI_NTV_SYNTAX,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: OCIStmtPrepare() crtstmt\n");
    report_error();
    return OCI_ERROR;
  }

  (void)fprintf (stdout, "\nCreating and populating table IMGDEMOTAB...\n");
  if (OCIStmtExecute(svchp,
                     stmthp,
                     errhp,
                     (ub4) 1,
                     (ub4) 0,
                     (CONST OCISnapshot *) 0,
                     (OCISnapshot *) 0,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: creating table\n");
    report_error();
    return OCI_ERROR;
  }

  /*
   * populate table with simple insert
   */
```

```
  if (OCIStmtPrepare(stmthp,
                     errhp,
                     insstmt,
                     (ub4) strlen((char *) insstmt),
                     (ub4) OCI_NTV_SYNTAX,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: OCIStmtPrepare() insstmt\n");
    report_error();
    return OCI_ERROR;
  }

  if (OCIStmtExecute(svchp,
                     stmthp,
                     errhp,
                     (ub4) 1,
                     (ub4) 0,
                     (CONST OCISnapshot *) 0,
                     (OCISnapshot *) 0,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: OCIStmtExecute() insstmt\n");
    report_error();
    return OCI_ERROR;
  }

  (void) OCITransCommit(svchp, errhp, (ub4)0);

  return OCI_SUCCESS;

}  /* end create_table */



/*
 *---------------------------------------------------------------------
 * fetch_blob - fetch the blob locator
 *---------------------------------------------------------------------
 */

sb4 fetch_blob()
{
  text  *sqlstmt = (text *)"SELECT A.C3.content FROM IMGDEMOTAB A";

  if (OCIStmtPrepare(stmthp,
```

```
                    errhp,
                    sqlstmt,
                    (ub4) strlen((char *)sqlstmt),
                    (ub4) OCI_NTV_SYNTAX,
                    (ub4) OCI_DEFAULT))
{
  (void) fprintf(stdout,"FAILED: OCIStmtPrepare() sqlstmt\n");
  report_error();
  return OCI_ERROR;
}

/*
 * define the output blob variable by position
 */
if (ocidefn(stmthp,
            defnp1,
            errhp,
            (ub4) 1,
            (dvoid *) &blob,
            (sb4) -1,
            (ub2) SQLT_BLOB,
            (dvoid *) 0,
            (ub2 *) 0,
            (ub2 *) 0,
            (ub4) OCI_DEFAULT))
{
  (void) fprintf(stdout,"FAILED: ocidefn()\n");
  report_error();
  return OCI_ERROR;
}

/* execute the select and fetch one row */
if (OCIStmtExecute(svchp,
                   stmthp,
                   errhp,
                   (ub4) 1,
                   (ub4) 0,
                   (CONST OCISnapshot*) 0,
                   (OCISnapshot*) 0,
                   (ub4) OCI_DEFAULT))
{
  (void) fprintf(stdout,"FAILED: OCIStmtExecute() sqlstmt\n");
  report_error();
  return OCI_ERROR;
}
```

```
  return OCI_SUCCESS;

}  /* fetch_blob */


/*
 *---------------------------------------------------------------------
 * drop_table - Drop table IMGDEMOTAB
 *---------------------------------------------------------------------
 */

void drop_table()
{
  text  *sqlstmt = (text *) "DROP TABLE IMGDEMOTAB";
  ub1   ebuf[256];
  text  *sqlstate=0;
  ub4   errcodep;

  (void)fprintf(stdout,"\nDropping table IMGDEMOTAB...\n");
  if (OCIStmtPrepare(stmthp,
                     errhp,
                     sqlstmt,
                     (ub4) strlen((char *) sqlstmt),
                     (ub4) OCI_NTV_SYNTAX,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: drop table\n");
    return;
  }

  (void)OCIStmtExecute(svchp,
                       stmthp,
                       errhp,
                       (ub4) 1,
                       (ub4) 0,
                       (CONST OCISnapshot *) 0,
                       (OCISnapshot *) 0,
                       (ub4) OCI_DEFAULT);
  return;

}  /* end drop_table */
```

```
/*
 * ---------------------------------------------------------------------
 * logout - Logoff and disconnect from the server.  Free handles.
 * ---------------------------------------------------------------------
 */

void logout()
{
  (void) OCISessionEnd(svchp, errhp, authp, (ub4) 0);
  (void) OCIServerDetach(srvhp, errhp, (ub4) OCI_DEFAULT);

  (void) fprintf(stdout,"\nLogged off and detached from server.\n");

  (void) OCIHandleFree((dvoid *) srvhp, (ub4) OCI_HTYPE_SERVER);
  (void) OCIHandleFree((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX);
  (void) OCIHandleFree((dvoid *) errhp, (ub4) OCI_HTYPE_ERROR);
  (void) OCIHandleFree((dvoid *) authp, (ub4) OCI_HTYPE_SESSION);
  (void) OCIDescriptorFree((dvoid *) blob, (ub4) OCI_DTYPE_LOB);
  (void) OCIDescriptorFree((dvoid *) bfile, (ub4) OCI_DTYPE_LOB);
  (void) OCIHandleFree((dvoid *) defnp1, (ub4) OCI_HTYPE_DEFINE);
  (void) OCIHandleFree((dvoid *) defnp2, (ub4) OCI_HTYPE_DEFINE);
  (void) OCIHandleFree((dvoid *) bndhp, (ub4) OCI_HTYPE_BIND);
  (void) OCIHandleFree((dvoid *) stmthp, (ub4) OCI_HTYPE_STMT);

  return;

}  /* end logout */


/*
 * ---------------------------------------------------------------------
 * report_error - retrieve error message and print it out.
 * ---------------------------------------------------------------------
 */
void report_error()
{
  text  msgbuf[1024];
  sb4   errcode;

  (void) OCIErrorGet((dvoid *) errhp,
                     (ub4) 1,
                     (text *) NULL,
                     &errcode,
                     msgbuf,
                     (ub4) sizeof(msgbuf),
```

```
                    (ub4) OCI_HTYPE_ERROR);
  (void) fprintf(stdout,"ERROR CODE = %d\n", errcode);
  (void) fprintf(stdout,"%s\n", msgbuf);
  return;

}  /* end report_error */


/*
 * ---------------------------------------------------------------
 * load_cart - load data into the image data cartridge
 *
 *   Populate the cartidge columns based on the contents of the bfile.
 *   The sequence of steps is to select the types and then use the
 *   contents to set the properties and perform the copy from the
 *   bfile to the blob.
 * ---------------------------------------------------------------
 */
sb4 load_cart()
{
text *sqlstmt = (text *)
"DECLARE \
  A ORDSYS.ORDImgF;\
  B ORDSYS.ORDImgB;\
BEGIN\
 SELECT C2, C3 INTO A,B FROM IMGDEMOTAB FOR UPDATE;\
\
 A.setProperties;\
 A.copyContent(B.content);\
 B.setProperties;\
 UPDATE IMGDEMOTAB SET C2 = A;\
 UPDATE IMGDEMOTAB SET C3 = B;\
 COMMIT;\
END;";

  if (OCIStmtPrepare(stmthp,
                     errhp,
                     sqlstmt,
                     (ub4) strlen((char *)sqlstmt),
                     (ub4) OCI_NTV_SYNTAX,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: OCIStmtPrepare() sqlstmt\n");
    report_error();
    return OCI_ERROR;
```

```
  }

  /* execute the select and fetch one row */
  if (OCIStmtExecute(svchp,
                     stmthp,
                     errhp,
                     (ub4) 1,
                     (ub4) 0,
                     (CONST OCISnapshot*) 0,
                     (OCISnapshot*) 0,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: OCIStmtExecute() sqlstmt\n");
    report_error();
    return OCI_ERROR;
  }

  return OCI_SUCCESS;

}  /* end load_cart */

/*
 * ---------------------------------------------------------------------
 * mod_img - modify some characteristics of the image
 *
 * This is accomplished by selecting the cartridge type and then
 * invoking the process method with a command.  The command specified
 * below scales the image by a factor of 2 and cuts a 100x100 pixel
 * piece from the original image.
 * ---------------------------------------------------------------------
 */
sb4 mod_img()
{
text *sqlstmt = (text *)
"DECLARE \
  B ORDSYS.ORDImgB;\
  command VARCHAR2(400);\
BEGIN\
 SELECT C3 INTO B FROM IMGDEMOTAB FOR UPDATE;\
\
 command := 'scale=2 cut=100 100 100 100';\
 B.process(command);\
 B.setProprties;\
 UPDATE IMGDEMOTAB SET C3 = B;\
 COMMIT;\
```

```
END;";

  if (OCIStmtPrepare(stmthp,
                     errhp,
                     sqlstmt,
                     (ub4) strlen((char *)sqlstmt),
                     (ub4) OCI_NTV_SYNTAX,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: OCIStmtPrepare() sqlstmt\n");
    report_error();
    return OCI_ERROR;
  }

  /* execute the select and fetch one row */
  if (OCIStmtExecute(svchp,
                     stmthp,
                     errhp, (ub4) 1,
                     (ub4) 0,
                     (CONST OCISnapshot*) 0,
                     (OCISnapshot*) 0,
                     (ub4) OCI_DEFAULT))
  {
    (void) fprintf(stdout,"FAILED: OCIStmtExecute() sqlstmt\n");
    report_error();
    return OCI_ERROR;
  }

  return OCI_SUCCESS;

}  /* end mod_img */

/*
 * ---------------------------------------------------------------------
 * blob_to_file - copy the contents of a blob to a file
 *
 * Read blob using stream mode into local buffers and then write
 * data to operating system file.
 * ---------------------------------------------------------------------
 */
sb4 blob_to_file()
{
  ub4   offset = 1;
  ub4   loblen = 0;
  ub1   bufp[MAXBUFLEN];
```

```
ub4   amtp = 0;
sword retval;
ub4   piece = 0;
ub4   remainder;
FILE  *fp;

/*
 * fetch the blob from which data will be read
 */
if (fetch_blob())
{
   (void)fprintf(stdout, "FAILED: fetch_blob()\n");
   return (OCI_ERROR);
}

fp = fopen((char *)fname, (const char *) "wb");

(void) OCILobGetLength(svchp, errhp, blob, &loblen);
amtp = loblen;

memset(bufp, '\0', MAXBUFLEN);

retval = OCILobRead(svchp,
                    errhp,
                    blob,
                    &amtp,
                    offset,
                    (dvoid *) bufp,
                    (loblen < MAXBUFLEN ? loblen : MAXBUFLEN),
                    (dvoid *)0,
                    (sb4 (*)(dvoid *, const dvoid *, ub4, ub1)) 0,
                    (ub2) 0,
                    (ub1) SQLCS_IMPLICIT);

switch (retval)
{
  case OCI_SUCCESS:             /* only one piece */
    (void) fwrite(bufp, loblen, 1, fp);
    break;
  case OCI_ERROR:
    report_error();
    break;
  case OCI_NEED_DATA:           /* there are 2 or more pieces */

    remainder = loblen;
```

```
    (void) fwrite(bufp, MAXBUFLEN, 1, fp); /* a full buffer to write */
    do
    {
      memset(bufp, '\0', MAXBUFLEN);
      amtp = 0;

      remainder -= MAXBUFLEN;

      retval = OCILobRead(svchp,
                          errhp,
                          blob,
                          &amtp,
                          offset,
                          (dvoid *) bufp,
                          (ub4) MAXBUFLEN,
                          (dvoid *)0,
                          (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                          (ub2) 0,
                          (ub1) SQLCS_IMPLICIT);

      /* the amount read returned is undefined for FIRST, NEXT pieces */

      if (remainder < MAXBUFLEN)     /* last piece not a full buffer piece */
         (void) fwrite(bufp, remainder, 1, fp);
      else
         (void) fwrite(bufp, MAXBUFLEN, 1, fp);

    } while (retval == OCI_NEED_DATA);
    break;
  default:
    (void) fprintf(stdout,"Unexpected ERROR: OCILobRead() LOB.\n");
    break;
}
if (ftell(fp) != loblen)
{
   (void)fprintf(
      stdout,
      "ERROR: resultant files size of %d differs from source lob size
      of &d\n",
      ftell(fp),loblen);
   status = OCI_ERROR;
 }

if (fclose(fp))
```

```
      return (OCI_ERROR);
   else
      return (status);

}  /* end blob_to_file */

/* end of file imgdemo.c */
```

C
Error Messages

IMG-00001, "unable to initialize Image Data Cartridge environment"

Cause: The image processing external procedure initialization process failed.

Action: Contact Oracle Worldwide Support.

IMG-00502, "invalid scale value"

Cause: An invalid scale value was found while parsing the parameters for the image process function.

Action: Correct the statement by using a valid scale value. Refer to the Image Cartridge documentation for a description of the correct usage and syntax for the image processing command string.

IMG-00505, "missing value in CUT rectangle"

Cause: An incorrect number of values was used to specify a rectangle.

Action: Use exactly four integer values for the lower left and upper right vertices.

IMG-00506, "extra value in CUT rectangle"

Cause: An incorrect number of values were used to specify a rectangle.

Action: Use exactly four integer values for the lower left and upper right vertices.

IMG-00510, application-specific-message

Cause: A syntax error was found while parsing the parameters for the image process function.

Action: Correct the statement by using valid parameter values. Refer to the Image Cartridge documentation for a description of the correct usage and syntax for the image processing command string.

IMG-00511, application-specific-message

Cause: An error was found while accessing image data.

Action: Contact Oracle Worldwide Support.

IMG-00531, "empty or null image processing command"

Cause: An empty or null image processing command was passed to the image process function.

Action: Refer to the Image Cartridge documentation for a description of the correct usage and syntax for the image processing command string.

IMG-00599, "internal error"

Cause: An internal error has occurred.

Action: Contact Oracle Worldwide Customer Support Services.

IMG-00601, "out of memory while copying image"

Cause: Operating system process memory has been exhausted while copying the image.

Action: See the database administrator or operating system administrator to increase process memory quota.

IMG-00602, "unable to access image data"

Cause: An error occurred while reading or writing image data.

Action: Contact your system administrator.

IMG-00603, "unable to access source image data"

Cause: The source image CONTENT attribute is invalid.

Action: Ensure that the CONTENT attribute of the source image is populated with image data.

IMG-00604, "unable to access destination image data"

Cause: The destination image CONTENT attribute is invalid.

Action: Ensure that the CONTENT attribute of the destination image is populated with a valid LOB locator.

IMG-00606, "unable to access image data"

Cause: An attempt was made to access an invalid image.

Action: Ensure that the CONTENT attribute of the image is populated with image data.

IMG-00607, "unable to write to destination image"

Cause: The destination image CONTENT attribute is invalid.

Action: Ensure that the CONTENT attribute of the destination image is populated with an initialized BLOB locator and that you have sufficient tablespace.

IMG-00609, "unable to read image stored in a BFILE"

Cause: The image stored in a BFILE cannot be opened for reading.

Action: Ensure that the access privileges of the image file and the image file's directory allow read access.

IMG-00701, "unable to set the properties of an empty image"

Cause: There is no data in the CONTENT attribute.

Action: Refer to the Image Cartridge documentation for information on how to populate image data into the CONTENT attribute of the ORDImgB or ORDImgF type.

IMG-00702, "unable to initialize image processing environment"

Cause: The image processing external procedure initialization process failed.

Action: Contact Oracle Worldwide Customer Support Services.

IMG-00703, "unable to read image data"

Cause: There is no image data in the CONTENT attribute.

Action: Refer to the Image Cartridge documentation for information on how to populate image data into the CONTENT attribute of the ORDImgB or ORDImgF type.

IMG-00704, "unable to read image data"

Cause: There is no image data in the CONTENT attribute.

Action: Refer to the Image Cartridge documentation for information on how to populate image data into the CONTENT attribute of the ORDImgB or ORDImgF type.

IMG-00705, "unsupported or corrupted input format"

Cause: This is an internal error.

Action: Contact Oracle Worldwide Customer Support Services.

IMG-00706, "unsupported or corrupted output format"

Cause: This is an internal error.

Action: Contact Oracle Worldwide Customer Support Services.

IMG-00707, "unable to access image data"

Cause: An error occurred while reading or writing image data.

Action: Contact your system administrator.

IMG-00710, "unable write to destination image"

Cause: The destination image is invalid.

Action: Ensure that the CONTENT attribute of the destination image is populated with an initialized BLOB locator and that you have sufficient tablespace.

IMG-00711, "unable to set properties of destination image"

Cause: This is an internal error.

Action: Contact Oracle Worldwide Customer Support Services.

IMG-00712, "unable to write to destination image"

Cause: The destination image is invalid.

Action: Ensure that the CONTENT attribute of the destination image is populated with an initialized BLOB locator and that you have sufficient tablespace.

IMG-00713, "unsupported destination image format"

Cause: A request was made to convert an image to a format that is not supported.

Action: Refer to the Oracle Image Cartridge Documentation for supported formats.

IMG-00714, "internal error"

Cause: This is an internal error.

Action: Contact Oracle Worldwide Customer Support Services.

IMG-00715, "Unable to open image stored in a BFILE"

Cause: The image stored in a BFILE could not be opened for reading.

Action: Ensure that the access privileges of the image file and the image file's directory allow read access.

Glossary

ADT

Abstract data type

BFILE

A large object whose value is composed of binary data, and is stored outside of the database in an operating system file. Because they are outside of the database, BFILEs are read-only.

binary large object

See BLOB.

BLOB

Binary large object. Large objects are stored in the database tablespace in a way that optimizes space and provide efficient access. Only a pointer to the object is actually stored in the row.

content format

A description of the image data, such as the pixel or color format.

cropping

Selecting the portion of an image within a specified rectangle and removing everything outside of that rectangle. *See* cutting.

cutting

Selecting the portion of an image within a specified rectangle to create a subimage. If the subimage is copied to a new image, the effect is a cut. If the subimage replaces the original, the effect is a crop. Use the process( ) or processCopy( ) procedures to cut or crop an image.

data cartridge

The mechanism by which clients, application-specific servers, and database servers can be easily and reliably extended.

default constructor

Using a default constructor in an OCI call creates an empty instance of the constructor.

file format

The file format of the image data, such as BMP or GIF.

IDL

Interface definition language

image

A graphic picture. The source could be a photograph, drawing, or generated image.

image processing

Changing the properties of an image, such as through scaling, rotation, or compression.

LOB

Large object. LOBs are used to hold large amounts of raw, binary data. Image Cartridge supports BLOB and BFILE LOBs.

lossless compression

A means for reducing the storage space required for an image. The decompressed image is bit-for-bit identical to the original.

lossy compression

A means for reducing the storage space required for an image. The decompressed image has less resolution than the original, although this might not be noticeable to the naked eye.

Network Computing Architecture

A fully integrated design integrating text, spatial, and image data. The architecture supports the design, development, installation, and integration of manageable components across entire organizations.

object relational database

A database having both object-oriented and relational characteristics. Objects can be defined and stored, and then retrieved using standard relational methods.

OCI

Oracle Call Interface

ODT

Object data type

scaling

Changing the proportions of an image in one or both dimensions. To enlarge an image, scale by a factor greater than one. To shrink an image, scale by a factor between zero and one. Use the process( ) or processCopy( ) procedures to scale an image.

Index

M

N

O

P

Q

R

S

T