Oracle® Visual Information Retrieval

User's Guide and Reference

Release 8.1.7

September 2000

Part No. A85335-01

Visual Information Retrieval technology licensed from Virage, Inc.

ORACLE®

Oracle Visual Information Retrieval User's Guide and Reference, Release 8.1.7

Part No. A85335-01

Primary Authors: Rod Ward and Mary Reinhardt

Contributors: Melliyal Annamalai, Dan Mullen, Sue Mavris, Rajiv Chopra, Deborah Owens, Brenda Silva

Contents

2 Content-Based Retrieval Concepts

3 Visual Information Retrieval Examples

4 Visual Information Retrieval Reference

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Visual Information Retrieval User's Guide and Reference, Release 8.1.7

Part No. A85335-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: nedc_doc@us.oracle.com
- FAX: 603.897.3316 Attn: Oracle Visual Information Retrieval Documentation
- Postal service:
  Oracle Corporation
  Oracle Visual Information Retrieval Documentation
  One Oracle Drive
  Nashua, NH 03062-2698
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This guide describes how to use Oracle Visual Information Retrieval. This product, based on technology licensed from Virage, Inc., extends the imaging capabilities of Oracle *inter*Media by allowing content-based retrieval of images.

Visual Information Retrieval requires Oracle8*i* Enterprise Edition.

For information about the differences between Oracle8*i* and Oracle8*i* Enterprise Edition and the features and options that are available to you, see *Getting to Know Oracle8i*.

Note: Since release 8.1.5, the ORDVIRB and ORDVIRF object types have been replaced by ORDVIR. Release 8.1.7 is the last release in which the old interface and object types will be available.

Audience

This guide is intended for anyone who is interested in storing, retrieving, and manipulating image data in an Oracle database, including developers of image specialization services.

Organization

This guide contains the following chapters and appendixes:

Chapter 1 Introduces image information retrieval and explains image-related concepts.

Chapter 2 Explains concepts, operations, and techniques related to content-based retrieval.

Chapter 3 Provides basic examples of using Visual Information Retrieval types and methods.

Chapter 4	Provides reference information on the Visual Information Retrieval object type, procedures, and operators.
Appendix A	Describes the supported image data formats.
Appendix B	Describes how to run the included sample application.
Appendix C	Describes in detail many of the parameters used to modify images.
Appendix D	Describes the raw pixel format used by foreign images.
Appendix E	Describes the deprecated interface used with a previous release of Visual Information Retrieval.
Appendix F	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 under your Oracle home directory. Depending on your operating system, this file may be in:

ORACLE_HOME`/ord/vir/admin/README.txt`

Please see your operating system-specific installation guide for more information.

Also see the Oracle Technology Network for the latest documentation:

`http://technet.oracle.com/`

For more information about using this product in a development environment, see the following documents in the Oracle8*i* documentation set:

- *Oracle Call Interface Programmer's Guide*
- *Oracle8i Application Developer's Guide - Large Objects (LOBs)*
- *Oracle8i Concepts*
- *PL/SQL User's Guide and Reference*

For information about the basic image storage and retrieval, see the *Oracle interMedia Audio, Image, and Video User's Guide and Reference*.

Visual Information Retrieval is based on technology licensed from Virage, Inc. Visit the Virage Web site for additional information at

`http://www.virage.com/`

Conventions

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.
. . .	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 text indicates a term defined in the text.
italic text	Italic text is used for emphasis, for user-supplied variables, and for book titles.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

Changes to This Guide

The following substantive changes were made to this guide since its previous version for release 8.1.5.

Other minor corrections and clarifications were also included.

Information is provided to ensure future compatibility of the 8.1.7 release with a future release of the evolving Visual Information Retrieval object type (ORDVir) containing new object attributes. Client-side applications should call the new compatibility initialization function (compatibilityInit( ) method) at the beginning of an application if necessary. See Section 4.1 for more information.

In addition, users are recommended to use the new static methods, init( ) and init(srcType, srcLocation,srcName); these two methods have been added to the

ORDVir media type to allow for easy initialization of instances of this type. Do not use the default constructors because INSERT statements using the default constructors will fail if the object type has evolved, adding new attributes. See Section 4.3 for more information.

The export( ) method now works for the source type FILE. See Section 4.4 for more information.

An additional ORDSource.import method has been defined. It is nearly identical to the existing import( ) method except that the destination BLOB is not passed in as a separate (redundant) parameter. See *Oracle interMedia Audio, Image, and Video User's Guide and Reference* for more information.

The deleteContent method no longer touches the metadata attributes. See Section 4.4 for more information.

The digital camera format known as EXIF is now recognized; it is a variation of the JFIF format, and the setProperties( ) method sets the fileFormat attribute to JFIF. See Table A–3 for more information.

1

Introduction

Oracle Visual Information Retrieval is an extension to Oracle8*i* Enterprise Edition that provides image storage, content-based retrieval, and format conversion capabilities through an object type. The capabilities of this product encompass the storage, retrieval, and manipulation of image data managed by the Oracle8*i* Enterprise Edition database server. This product supports image storage using binary large objects (BLOBs) and references to image data residing externally in BFILEs or URLs.

Visual Information Retrieval is a building block for various imaging applications, rather than being an end-user application in itself. It consists of an object type along with related methods for managing and processing image data. Some example applications are:

- Online digital art galleries or museums
- Real estate marketing
- Stock photograph collections (for example, for fashion designers or architects)

These applications have certain distinct requirements and some degree of commonality. The image object type accommodates the commonality and supports extensions that address application-specific requirements. With Visual Information Retrieval, images can be managed as easily as standard attribute data.

Visual Information Retrieval supports static, two-dimensional images in Oracle databases. The images may be bitonal (black and white) images, grayscale photographs, or color photographic images. Certain popular image formats are natively understood by Visual Infromation Retrieval. For these supported image formats, Visual Information Retrieval can automatically extract properties, convert to other formats and compression schemes, and cut and scale the image. Any image format can be stored in the object relational type known as ORDVir, associated with other column data and retrieved from ORDVir. This enables database designers to

extend existing database applications with images or to build new end-user image database applications. Software developers can use the basic functions provided here to build specialized image applications.

1.1 If You Already Understand Oracle *inter*Media Image

If you are already familiar with Oracle *inter*Media Image service, either as a component of Oracle *inter*Media or in a previous release as Oracle8 Image Cartridge, you can skim much of the conceptual information in this chapter. The base Image component and Visual Information Retrieval both let you store an image as an object in the database or as a reference to an external file or URL. Both products let you store and query on the following attributes:

- Image height
- Image width
- Image size
- File type or format (such as TIFF)
- Compression type or format (such as JPEG)
- Content type (such as monochrome)
- MIME type

The Visual Information Retrieval object type is defined as the Image object, plus a signature attribute.

The main differences between the Image component and the Visual Information Retrieval product are that Visual Information Retrieval lets you create and use indexes, and perform **content-based retrieval**. Content-based retrieval lets you perform queries based on intrinsic visual attributes of the image (color, structure, texture), rather than being limited to keyword searches in textual annotations or descriptions. The underlying technology was developed by Virage, Inc., a leader in content-based retrieval.

For an example of a query using content-based retrieval, consider a database containing images of many automobiles. If you want to retrieve information on all the red automobiles, you would specify an image of a red automobile for comparison and request all records where the image looks like your picture. To increase the accuracy of the query (because all the images are of automobiles and you are interested only in red ones), you specify that the greatest relative weight is to be given to the global color attribute, with no weight given to the structure and texture attributes.

For further information on content-based retrieval, including how and why to specify relative weights (importance) for different visual attributes, see Chapter 2.

1.2 Image Concepts

This section contains conceptual material about digital images. Chapter 2 contains conceptual information about content-based retrieval and using Visual Information Retrieval to build image applications or specialized image services.

1.2.1 Digital Images

Visual Information Retrieval 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 video tape recorder 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.

Visual Information Retrieval provides the mechanism to integrate the storage and retrieval of images in Oracle databases using the Oracle8*i* Enterprise Edition database server.

1.2.2 Image Components

A digital image can be thought of as consisting of the image data (digitized bits) and attributes that describe the characteristics of the image. Image applications sometimes associate application-specific information, such as the name of the person whose image a photograph represents, with an image by storing descriptive text in an attribute or column in the database table.

The minimal attributes carried along with 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 that were sampled. The data attributes describe the image as it was produced by the capture device.

The image data (pixels) can have varying depths (bits per pixel) depending upon how the image was captured, and the image data can be organized in various ways. The organization of the image data, known as the data format, is crucial to accessing and accurately representing the image.

The size of digital images (number of bytes) tends to be large compared to traditional computer objects such as numbers and text. Therefore, many compression schemes are in use that squeeze an image into fewer bytes, thus putting a smaller load on storage devices and networks. *Lossless* compression schemes squeeze an image in such a fashion that when it is decompressed, the resulting image is bit-for-bit identical to the original. *Lossy* compression schemes do not result in a bit-for-bit identical image when decompressed, but the differences may be imperceptible to the human eye, or at worst, tolerable.

Visual Information Retrieval also provides a signature attribute in the image type that permits content-based retrieval. The **signature** is a vector containing detailed information about the visual attributes of the image. The signature is created when the image is processed by Visual Information Retrieval, and is used in all content-based queries. For more information about the signature attribute, see Section 2.2.

1.2.3 Interchange Formats

An **image interchange format** describes a well-defined organization and use of image attributes, data, and often a compression scheme, allowing different applications to create, interchange, and use images. Interchange formats are often stored in or as disk files, but 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 there are many applications that create or use digital images within these. To assist application developers, Visual Information Retrieval supports many popular interchange formats. (See Appendix A, "File and Compression Formats".)

1.3 Object Relational Technology

Oracle8*i* 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 provides support for the definition of object types including the data associated with an object (attributes) and the operations that can be performed on it (methods). This powerful mechanism, well established in the object-oriented world, includes integral support for binary large objects (BLOBs) to provide the basis for adding complex objects, such as digital images, to Oracle8*i* databases.

See the following for extensive information on using BLOBs and BFILEs:

- *Oracle8i Application Developer's Guide - Large Objects (LOBs)*
- *Oracle8i Concepts*

1.3.1 Storing Images

Visual Information Retrieval can store digital images within the Oracle database under transactional control through the BLOB mechanism. It can also externally reference digital images stored in flat files through the BFILE mechanism, or an HTTP server-based URL. Although this is particularly convenient for integrating pre-existing sets of flat-file images with an Oracle database, these images will not be under transactional control.

The object relational type is known as ORDVir, and is based on the ORDImage object type, which in turn is based on the ORDSource object type. See the *Oracle interMedia Audio, Image, and Video User's Guide and Reference* for more details on ORDSource and ORDImage.

1.3.2 Querying Images

Once stored within an Oracle database, an image can be found using traditional queries by finding a row in a table that contains the image using the various alphanumeric columns of the table. For example, select a photograph from the automobile table where the automobile name is Packard. An example of content-based retrieval might be as follows: compare a picture of a parked automobile in the parking lot to the stored photographs in the automobile table, retrieve the most similar image, and check to see which employee is the owner.

The meaning of "similar" can be refined in image comparisons. You can experiment with different weight values for the visual attributes of global color, local color, texture, and structure.

The collection of digital images in the database can be related to some set of attributes or keywords that describe the associated content. The image content can be described with text and numbers such as dates and identification numbers. For Oracle, image attributes can reside in the same table as the image object type. Alternatively, an application designer could define a composite object type that contains the Visual Information Retrieval object type along with other attributes.

1.3.3 Accessing Images

Applications access and manipulate images using SQL, PL/SQL, or Java through the ORDVir image object type. 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, ...)

See *Oracle8i Concepts* for information on this and other SQL syntax.

See *Oracle Visual Information Retrieval Java Classes User's Guide and Reference* for information on the Visual Information Retrieval Java interface.

1.4 Visual Information Retrieval Methods and Operations

Visual Information Retrieval provides several functions for performing format conversion, compression, and data manipulation operations on image data. It also provides the ability to extract image properties and to compare images based on their content. This section presents a conceptual overview of the Visual Information Retrieval methods and operations.

1.4.1 Analyzing and Comparing Images

The Analyze( ) operator and analyze method examine an image and create a signature based on the global and local colors, texture, and structure of the image content. Two additional operators, VIRScore( ) and VIRSimilar( ), compare the signatures of two images to determine if the images match based on a set of user-supplied criteria.

1.4.2 Extracting Properties from Images

The setProperties( ) method is used to extract important properties from natively supported image data, including the following:

- Height and width in pixels
- Total size in bytes
- File format (such as TIFF or BMP)
- Compression format (such as JPEG or LZW)
- Content format (such as monochrome or 8-bit grayscale)
- MIME type

The size of the image data may be machine dependent. Importing and exporting images may require a recalculation using the setProperties( ) method to determine the current properties.

A set of functions prefaced with "get" can be used to retrieve individually stored attributes of natively supported images. For example, getHeight( ) and getWidth( ) return the image height and width in pixels. See Chapter 4 for a complete list of these functions and their syntax.

1.4.3 Verifying Image Properties

The checkProperties( ) method is used to verify that the properties stored in attributes of an image object match the actual image properties. This function operates on any natively supported image format.

1.4.4 Modifying Images

The process( ) and processCopy( ) methods are used for image format conversion, compression, and basic manipulation functions including scaling and cutting. The image may be compressed using an algorithm from the set of supported image formats and compression schemes. For example, image data in the TIFF format may be compressed using Packbits, Huffman, JPEG, LZW, or one of the other supported schemes. Appendix A, "File and Compression Formats", lists the supported image formats and related compression schemes. Appendix C, "Process and ProcessCopy Operators", describes the characteristics that can be modified.

For this release, the output of any image manipulation function must be directed to a BLOB. In-place modification is supported for BLOBs, not for BFILEs or URLs.

1.4.5 Moving Images

There are several ways to move images between systems, databases, or records:

- The copy( ) and processCopy( ) methods copy an image into another ORDVir object.
- The export( ), import( ), and importFrom( ) methods are useful in moving images between the database and external data sources. The Convert( ) operator can be used to convert the image signature if you are moving images between different hardware platforms.
- The setSource( ) method sets or alters information about an image stored externally.

1.4.6 Deleting Images

The deleteContent( ) method allows you to remove the contents of the image BLOB.

Because external files are read-only, this method works only with images stored in BLOBs.

1.4.7 Setting Image Characteristics Manually

Foreign images are images not natively understood by Visual Information Retrieval. Although any image can be stored in an ORDVir object, the characteristics of foreign images cannot be read from the image header automatically. A special version of the setProperties( ) method exists to let you set these characteristics explicitly. Note that there is no verification that you have set the characteristics appropriately.

The setUpdateTime( ) method is called automatically, whenever native images are modified. You must call this method explicitly for foreign images.

2

Content-Based Retrieval Concepts

This chapter explains, at a high level, why and how to use content-based retrieval. It covers the following topics:

- Overview and benefits of content-based retrieval
- How content-based retrieval works, including definitions and explanations of the visual attributes (global color, local color, texture, structure) and why you might emphasize specific attributes in certain situations
- Image matching using a specified comparison image, including comparing how the weights of visual attributes determine the degree of similarity between images
- Image preparation or selection to maximize the usefulness of comparisons
- Use of indexing to improve search and retrieval performance

2.1 Overview and Benefits

Inexpensive image-capture and storage technologies have allowed massive collections of digital images to be created. However, as a database grows, the difficulty of finding relevant images increases. Two general approaches to this problem have been developed, both of which use metadata for image retrieval:

- Using information manually entered or included in the table design, such as titles, descriptive keywords from a limited vocabulary, and predetermined classification schemes
- Using automated image feature extraction and object recognition to classify image content -- that is, using capabilities unique to content-based retrieval

With Visual Information Retrieval, you can combine both approaches in designing a table to accommodate images: use traditional text columns to describe the semantic

significance of the image (for example, that the pictured automobile won a particular award, or that its engine has six or eight cylinders), and use the Visual Information Retrieval type for the image, to permit content-based queries based on intrinsic attributes of the image (for example, how closely its color and shape match a picture of a specific automobile).

As an alternative to defining image-related attributes in columns separate from the image, a database designer could create a specialized composite data type that combines Visual Information Retrieval and the appropriate text, numeric, and date attributes.

The primary benefit of using content-based retrieval is reduced time and effort required to obtain image-based information. With frequent adding and updating of images in massive databases, it is often not practical to require manual entry of all attributes that might be needed for queries, and content-based retrieval provides increased flexibility and practical value. It is also useful in providing the ability to query on attributes such as texture or structure that are difficult to represent using keywords.

Examples of database applications where content-based retrieval is useful -- where the query is semantically of the form, “find objects that look like this one” -- include:

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

For example, a Web-based interface to a retail clothing catalog might allow users to search by traditional categories (such as style or price range) and also by image properties (such as color or texture). Thus, a user might ask for formal shirts in a particular price range that are off-white with pin stripes. Similarly, fashion designers could use a database with images of fabric swatches, designs, concept sketches, and finished garments to facilitate their creative processes.

2.2 How Content-Based Retrieval Works

A content-based retrieval system processes the information contained in image data and creates an abstraction of its content in terms of visual attributes. Any query operations deal solely with this abstraction rather than with the image itself. Thus,

every image inserted into the database is analyzed, and a compact representation of its content is stored in a feature vector, or **signature**.

The signature contains information about the following visual attributes:

- **Global color** represents the distribution of colors within the entire image. This distribution includes the amounts of each color, but not the locations of colors.
- **Local color** represents color distributions *and where they occur* in an image, such as the fact that a red-green-blue (RGB) vector for sky blue occurs in the upper half of an image.
- **Texture** represents the low-level patterns and textures within the image, such as graininess or smoothness. Unlike structure, texture is very sensitive to features that appear with great frequency in the image.
- **Structure** represents the shapes that appear in the image, as determined by shape-characterization techniques such as edge detection.

Feature data for all these visual attributes is stored in the signature, whose size typically ranges from 1000 to 2000 bytes. For better performance with large image databases, you can create an index based on the signatures of your images. See Section 2.4 for more information on indexing.

Images in the database can be retrieved by matching them with a comparison image. The comparison image can be any image inside or outside the current database, a sketch, an algorithmically generated image, and so forth.

The matching process requires that signatures be generated for the comparison image and each image to be compared with it. Images are seldom identical, and therefore matching is based on a similarity-measuring function for the visual attributes and a set of weights for each attribute. The **score** is the relative distance between two images being compared. The score for each attribute is used to determine the degree of similarity when images are compared, with a smaller distance reflecting a closer match, as explained in Section 2.3.3.

2.2.1 Global Color and Local Color

Global color reflects the distribution of colors within the entire image, whereas local color reflects color distributions *and where they occur* in an image. To illustrate the difference between global color and local color, consider Figure 2–1.

Figure 2–1 Image Comparison: Global Color and Local Color

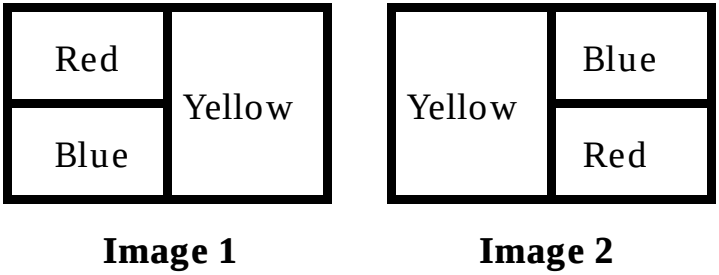


Image 1 and Image 2 are the same size and are filled with solid colors. In Image 1, the top left quarter (25%) is red, the bottom left quarter (25%) is blue, and the right half (50%) is yellow. In Image 2, the top right quarter (25%) is blue, the bottom right quarter (25%) is red, and the left half (50%) is yellow.

If the two images are compared first solely on global color and then solely on local color, the following are the similarity results:

- Global color: complete similarity (score = 0.0), because each color (red, blue, yellow) occupies the same percentage of the total image in each one
- Local color: no similarity (score = 100), because there is no overlap in the placement of any of the colors between the two images

Thus, if you need to select images based on the dominant color or colors (for example, to find apartments with blue interiors), give greater relative weight to global color. If you need to find images with common colors in common locations (for example, red dominant in the upper portion to find sunsets), give greater relative weight to local color.

Figure 2–2 shows two images very close (score = 0.0) in global color. Figure 2–3 shows two images very close (score = 0.02461) in local color.

Figure 2–2 Images Very Similar in Global Color

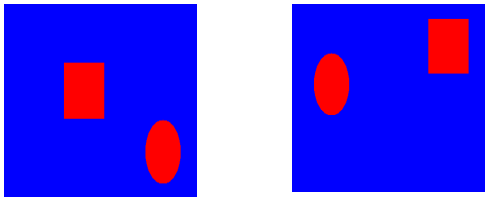
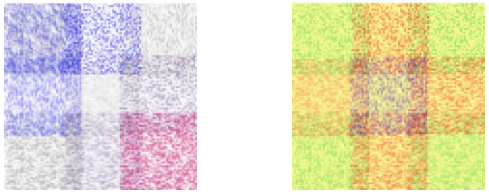


Figure 2–3 Images Very Similar in Local Color

2.2.2 Texture and Structure

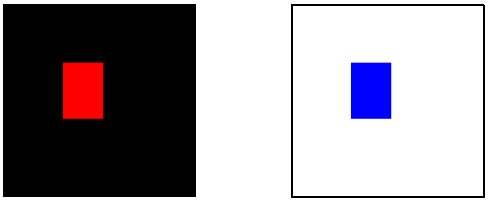
Texture is most useful for full images of textures, such as catalogs of wood grains, marble, sand, or stones. These images are generally hard to categorize using keywords alone because our vocabulary for textures is limited. Texture can be used effectively alone (without color) for pure textures, but also with a little bit of global color for some kinds of textures, like wood or fabrics. Figure 2–4 shows two similar fabric samples (score = 4.1).

Figure 2–4 Fabric Images with Similar Texture



Structure is not strictly confined to certain sizes or positions. However, when objects are of the same size or position, they have a lower score (greater similarity) than objects of different sizes. Structure is useful to capture objects such as horizon lines in landscapes, rectangular structures in buildings, and organic structures like trees. Structure is very useful for querying on simple shapes (like circles, polygons, or diagonal lines) especially when the query image is drawn by hand and color is not considered important when the drawing is made. Figure 2–5 shows two images very close (score = 0.61939) in structure.

Figure 2–5 Images with Very Similar Structure



2.3 How Matching Works

When you match images, you assign an importance measure, or weight, to each of the visual attributes, and Visual Information Retrieval calculates a similarity measure for each visual attribute.

2.3.1 Weight

Each **weight** value reflects how sensitive the matching process for a given attribute should be to the degree of similarity or dissimilarity between two images. For example, if you want global color to be completely ignored in matching, assign a weight of 0.0 to global color; in this case, any similarity or difference between the global color of the two images is totally irrelevant in matching. On the other hand, if global color is extremely important, assign it a weight greater than any of the other attributes; this will cause any similarity or dissimilarity between the two images with respect to global color to contribute greatly to whether or not the two images match.

Weight values can be any positive real numbers. During processing, the values are normalized such that they sum to 100%. To avoid confusion, you should get into a habit of always using the same scale (0.00 to 1.00, or 0 to 100) when providing

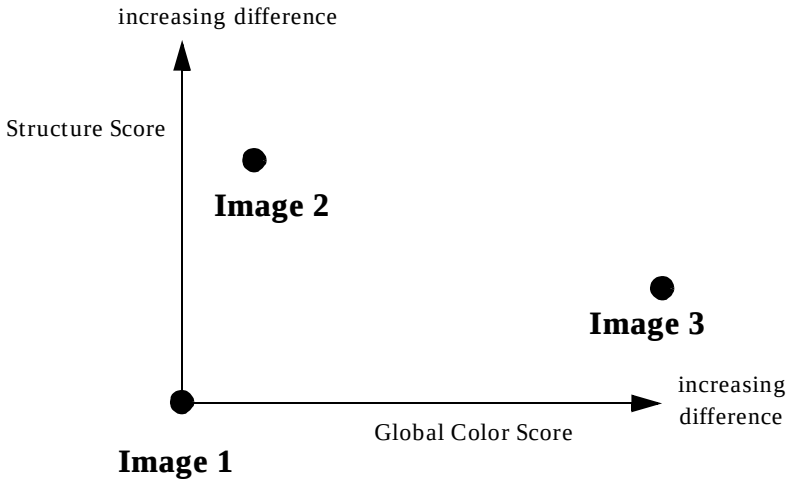
weight values. At least one of the visual attribute weights must be set greater than zero. See Section 2.3.3 for details of the calculation.

2.3.2 Score

The similarity measure for each visual attribute is calculated as the **score** or distance between the two images with respect to that attribute. The score can range from 0 (no difference) to 100 (maximum possible difference). Thus, the more similar two images are with respect to a visual attribute, the *smaller* the score will be for that attribute.

As an example of how distance is determined, assume that the dots in Figure 2–6 represent scores for three images with respect to two visual attributes, such as global color and structure, plotted along the x-axis and y-axis of a graph.

Figure 2–6 Score and Distance Relationship



For matching, assume Image 1 is the comparison image, and Image 2 and Image 3 are each being compared with Image 1. With respect to the global color attribute plotted on the x-axis, the distance between Image 1 and Image 2 is relatively small (for example, 15), whereas the distance between Image 1 and Image 3 is much greater (for example, 75). If the global color attribute is given more weight, then the fact that the two distance values differ by a great deal will probably be very important in determining whether or not Image 2 and Image 3 match Image 1. However, if global color is minimized and the structure attribute is emphasized instead, then Image 3 will match Image 1 better than Image 2 matches Image 1.

2.3.3 Similarity Calculation

In Section 2.3.2, Figure 2–6 showed a graph of only two of the attributes that Visual Information Retrieval can consider. In reality, when images are matched, the degree of similarity depends on a weighted sum reflecting the weight and distance of all four of the visual attributes of the comparison image and the test image.

For example, assume that for the comparison image (Image 1) and one of the images being tested for matching (Image 2), Table 2–1 lists the relative distances between the two images for each attribute. Note that you would never see these individual numbers unless you computed four separate scores, each time highlighting one attribute and setting the others to zero.

Table 2–1 Distances for Visual Attributes Between Image1 and Image2

Visual Attribute	Distance
Global color	15
Local color	90
Texture	5
Structure	50

In this example, the two images are most similar with respect to texture (distance = 5) and most different with respect to local color (distance = 90).

Assume that for the matching process, the following weights have been assigned to each visual attribute:

- Global color = 0.1
- Local color = 0.6
- Texture = 0.2
- Structure = 0.1

The weights are typically supplied in the range of 0.0 to 1.0. Within this range, a weight of 1 indicates the strongest emphasis, and a weight of 0 means the attribute should be ignored. You can use a different range (such as 0 to 100), but be careful not to accidentally combine different ranges. The values you supply are automatically normalized such that the weights total 100 percent, still maintaining the ratios you have supplied. In this example, the weights were specified such that normalization was not necessary.

The following formula is used to calculate the weighted sum of the distances, which is used to determine the degree of similarity between two images:

```
weighted_sum = global_color_weight * global_color_distance +
               local_color_weight  * local_color_distance +
               texture_weight      * texture_distance +
               structure_weight    * structure_distance
```

The degree of similarity between two images in this case is computed as:

0.1*gc_distance + 0.6*lc_distance + 0.2*tex_distance + 0.1*struc_distance

Using the supplied values, this becomes:

(0.1*15 + 0.6*90 + 0.2*5 + 0.1*50) = (1.5 + 54.0 + 1.0 + 5.0) = 61.5

To illustrate the effect of different weights in this case, assume that the weights for global color and local color were reversed. In this case, the degree of similarity between two images is computed as:

0.6*gc_distance + 0.1*lc_distance +0.2*tex_distance + 0.1*struc_distance

That is:

(0.6*15 + 0.1*90 + 0.2*5 + 0.1*50) = (9.0 +9.0 + 1.0 + 5.0) = 24.0

In this second case, the images are considered to be more similar than in the first case, because the overall score (24.0) is smaller than in the first case (61.5). Whether or not the two images are considered matching depends on the threshold value (explained in Section 2.3.4). If the weighted sum is less than or equal to the threshold, the images match; if the weighted sum is greater than the threshold, the images do not match.

In these two cases, the *correct* weight assignments depend on what you are looking for in the images. If local color is extremely important, then the first set of weights is a better choice than the second set of weights, because the first set of weights grants greater significance to the disparity between these two specific images with respect to local color (weighted sum of 24 versus 61.5). Thus, with the first set of weights, these two images are less likely to match -- and a key goal of content-based retrieval is to eliminate unimportant images so that you can focus on images containing what you are looking for.

2.3.4 Threshold Value

When you match images, you assign a **threshold** value. If the weighted sum of the distances for the visual attributes is less than or equal to the threshold, the images match; if the weighted sum is greater than the threshold, the images do not match.

Using the examples in Section 2.3.3, if you assign a threshold of 50, the images do not match when the weighted sum is 61.5, but they do match when the weighted sum is 24. If the threshold is 20, the images do not match in either case; and if the threshold is 61.5 or greater, the images match in both cases.

The following example shows a cursor (getphotos) that selects the photo_id, annotation, and photo from the Pictures table where the threshold value is 20 for comparing photographs with a comparison image:

```
CURSOR getphotos IS
 SELECT photo_id, annotation, photo FROM Pictures WHERE
   ORDSYS.VIRSimilar(photo.Signature, comparison_sig, 'globalcolor="0.4",
   localcolor="0.2", texture="0.10", structure="0.3"', 20)=1;
```

Before the cursor executes, the Analyze( ) operator must be used to compute the signature of the comparison image (comparison_sig), and to compute signatures for each image in the table. Chapter 4 describes all the operators, including Analyze( ) and Similar( ).

The number of matches returned generally increases as the threshold increases. Setting the threshold to 100 would return all images as matches. Such a result, of course, defeats the purpose of content-based retrieval. If your images are all very similar, you may find that even a threshold of 50 returns too many (or all) images as matches. Through trial and error, adjust the threshold to an appropriate value for your application.

You will probably want to experiment with different weights for the visual attributes and different threshold values, to see which combinations retrieve the kinds and approximate number of matches you want.

2.4 Using an Index to Compare Signatures

A domain index, or extensible index, is a new approach for supporting complex data objects. Oracle and Visual Information Retrieval cooperate to define, build, and maintain an index for image data. This index is of type ORDVIRIDX. Once it is created, the index automatically updates itself every time an image is inserted or removed from the database table. The index is created, managed, and accessed by routines supplied by the index type.

For better performance with large image databases, you should always create and use an index for searching through the image signatures. The default search model compares the signature of the query image to the signatures of all images stored in the database. This works well for simple queries against a few images such as, "Does this picture of an automobile match the image stored with the client's insurance records?" However, if you want to compare that image with thousands or millions of images to determine what kind of vehicle it is, then a linear search though the database would be impractical. In this case, an index based on the image signatures would greatly improve performance.

Assume you have a table T containing fabric ID numbers and pattern photographs:

```
CREATE TABLE T (fabric_id NUMBER, pattern_photo ORDSYS.ORDVIR);
```

Load the table with images, and process each image using the Analyze( ) operator to generate the signatures.

Note: Performance is greatly improved by loading the data tables prior to creating the index.

Once the signatures are created, the following command creates an index on this table, based on the data in the `pattern_photo` column.

```
CREATE INDEX idx1 ON T(pattern_photo.signature) INDEXTYPE IS ORDSYS.ORDVIRIDX
      PARAMETERS ('ORDVIR_DATA_TABLESPACE = <name>,ORDVIR_INDEX_TABLESPACE = <name>');
```

The index name is limited to 24 or fewer characters.[1] As with any Oracle table, do not use pound signs (#) or dollar signs ($) in the name. Also as usual, the tablespace must be created before creating the table.

The index data resides in two tablespaces. The first contains the actual index data, and the second is an internal index created on that data. See Section 3.8 for suggestions concerning the sizes of these tablespaces.

Finally, as with other Oracle indexes, you should analyze the new index as follows:

```
ANALYZE INDEX idx1 COMPUTE STATISTICS;
```

[1] The standard Oracle restriction is 30 characters for table or index names. However, Visual Information Retrieval requires an extra 6 characters for internal processing of the domain index.

Two operators, VIRSimilar( ) and VIRScore( ) support queries using the index. The operators automatically use the index if it is present. See Section 4.5 for syntax information and examples.

2.5 Preparing or Selecting Images for Useful Matching

The human mind is infinitely smarter than a computer in matching images. If we are near a street and want to identify all red automobiles, we can easily do so because our minds rapidly adjust for the following factors:

- Whether the automobile is stopped or moving
- The distinction between red automobiles, red motorcycles, and red trailers
- The absolute size of the automobile, as well as its relative size in our field of vision (because of its distance from us)
- The location of the automobile in our field of vision (center, left, right, top, bottom)
- The direction in which the automobile is pointing or traveling (left or right, toward us, or away from us)

However, for a computer to find red automobiles (retrieving all red automobiles and no or very few images that are not red or not automobiles), it is helpful if all the automobile images have the automobile occupy almost the entire image, have no extraneous elements (people, plants, decorations, and so on), and have the automobiles pointing in the same direction. In this case, a match emphasizing global color and structure would produce useful results. However, if the pictures show automobiles in different locations, with different relative sizes in the image, pointing in different directions, and with different backgrounds, it will be difficult to perform content-based retrieval with these images.

The following are some suggestions for selecting images or preparing images for comparison. The list is not exhaustive, but the basic principle to keep in mind is this: Know what you are looking for, and use common sense. If possible, crop and edit images in accordance with the following suggestions before performing content-based retrieval:

- Have what you expect to be looking for occupy almost all the image space, or at least occupy the same size and position on each image. For example, if you want to find all the red automobiles, each automobile image should show only the automobile and should have the automobile in approximately the same position within the overall image.

- Minimize any extraneous elements that might prevent desired matches or cause unwanted matches. For example, if you want to match red automobiles and if each automobile has a person standing in front of it, the color, shape, and position of the person (skin and clothing) will cause color and shape similarities to be detected, and might reduce the importance of color and shape similarities between automobiles (because part of the automobile is behind the person and thus not visible). If you know that your images vary in this way, experiment with different thresholds and different weights for the various visual attributes until you find a combination that provides the best result set for your needs.
- During analysis, images are temporarily scaled to a common size such that the resulting signatures are based on a common frame of reference. If you crop a section of an image, and then compare that piece back to the original, Visual Information Retrieval will likely find that the images are less similar than you would expect.

Note: Visual Information Retrieval operates as a fuzzy search engine, and is not designed to do correlations. For example, Visual Information Retrieval cannot find a specific automobile in a parking lot. However, if you crop an individual automobile from a picture of a parking lot, you can then compare the automobile to known automobile images.

3

Visual Information Retrieval Examples

This chapter provides examples of common operations with Visual Information Retrieval. These operations include:

- Creating tablespaces, creating a user, granting user privileges, and creating a directory
- Creating a new table with an image column (based on the ORDVir object type)
- Modifying an existing table to add an image column
- Loading images into the table (includes setting the image properties and generating the signatures) using PL/SQL
- Loading images into a table using SQL*Loader and using a PL/SQL program to set the properties and generate the signatures
- Retrieving images (simple read operation; no content-based retrieval)
- Retrieving images similar to a comparison image (content-based retrieval)
- Creating a domain index
- Retrieving images similar to a comparison image using indexing operations (indexed content-based retrieval)
- Converting an image from one format to another format
- Extending the product with new object types
- Using object views
- Using a set of scripts for creating and loading an image table from a BFILE data source

The examples in this chapter use a table of photographs. For each photograph, a photo ID, the photographer's name, a descriptive annotation, and the photographic image are stored.

Reference information on the functions used in these examples is presented in Chapter 4.

3.1 Creating Tablespaces, Creating a User, Granting User Privileges, and Creating a Directory

Example 3–1 contains a script that creates the tablespaces needed for some of the examples in this chapter, creates a vir1 user with password vir1, identifies the default and temporary tablespace for this user's sessions, grants the necessary privileges to the vir1 user, and then connects as vir1 user and creates the *ordvirdir* directory.

> **Note:** The following examples assume tablespace named temp has been created. Tablespace temp is not created by default.

Example 3–1 Create Tablespaces, a User, Grant User Privileges, and Create a Directory

```
CONNECT system/<system-password>;

SET SERVEROUTPUT ON
SET ECHO ON

-- create tablespace tbs_1
CREATE TABLESPACE tbs_1
       DATAFILE 'tbs_1.dbf' SIZE 20M
       MINIMUM EXTENT 64K
       DEFAULT STORAGE (INITIAL 64K NEXT 128K)
       LOGGING;

-- create tablespace tbs_2
CREATE TABLESPACE tbs_2
       DATAFILE 'tbs_2.dbf' SIZE 20M
       MINIMUM EXTENT 64K
       DEFAULT STORAGE (INITIAL 64K NEXT 128K)
       LOGGING;

CREATE USER user vir1 IDENTIFIED BY vir1
DEFAULT TABLESPACE tbs_1
```

```
TEMPORARY TABLESPACE temp;

GRANT CONNECT, RESOURCE, CREATE LIBRARY TO vir1;
GRANT CREATE ANY DIRECTORY TO vir1;

CONNECT vir1/vir1;
CREATE OR REPLACE DIRECTORY ordvirdir
       as 'e:\oracle\ord\vir\demo';
GRANT READ ON DIRECTORY ordvirdir TO PUBLIC WITH GRANT OPTION;
```

Note: If the *ordvirdir* directory will be used by any other schema, you must grant read access privilege to it as shown in the last line of Example 3–1.

3.2 Creating a New Table Containing an Image Column

Example 3–2 creates a new table of photographic images that includes the following information for each photograph:

- Photo ID number
- Photographer's name (variable text, up to 64 characters)
- Descriptive annotation (variable text, up to 255 characters)
- Photographic image (defined using Visual Information Retrieval)

Example 3–2 Create a New Table Containing an Image

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON

CREATE TABLE stockphotos (photo_id NUMBER, photographer VARCHAR2(64),
            annotation VARCHAR2(255), photo ORDSYS.ORDVir);
```

The SQL DESCRIBE table statement shows the following description:

```
DESCRIBE stockphotos;
Column Name                    Null?    Type
------------------------------ -------- ----
PHOTO_ID                                NUMBER
```

```
PHOTOGRAPHER                    VARCHAR2(64)
ANNOTATION                      VARCHAR2(255)
PHOTO                           ORDSYS.ORDVIR
```

3.3 Adding an Image Column to an Existing Table

This example modifies an existing table to store an image with each row. Assume that the table in Section 3.2 already exists, but the table does not include the actual photographic images (the photo column).

Example 3–3 adds the `photo` column to the stockphotos table. Later, when the images are loaded, they can be stored either in the database itself or as references to external files.

Example 3–3 Add an Image Column to an Existing Table

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON
ALTER TABLE stockphotos ADD (photo ORDSYS.ORDVir);
```

3.4 Loading an Image from an External File

Example 3–4 shows how to load two external images and import them into the database. This example assumes the image files are located in the *ordvirdir* directory. This directory name must be specified in uppercase in the INSERT statement.

Example 3–4 Load an Image into a Table

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON

DECLARE
   image     ORDSYS.ORDVIR;
   IdNum     NUMBER;
   ctx       RAW(4000) :=NULL;
BEGIN
   -- Generate a photo ID and insert a row into the table.
   -- Note: empty_blob() is the initializer for the BLOB attribute.
   IdNum := 1;
   INSERT INTO stockphotos VALUES (IdNum, 'Janice Gray',
          'Beach scene, balls on water',
          ORDSYS.ORDVIR.init('FILE','ORDVIRDIR','virdemo1.dat));
```

```
   SELECT photo INTO image FROM stockphotos WHERE photo_id = IdNum FOR UPDATE;

   -- Import calls setProperties by default.
   -- Read the data into the database and set properties.
      image.import(ctx);

   -- Generate the image signature.
      image.Analyze;

   -- Update the photo column with the contents of image.
   -- This also stores the signature and other image-specific attributes.
   UPDATE stockphotos SET photo = image WHERE photo_id = IdNum;
   COMMIT;

   IdNum := 2;
   INSERT INTO stockphotos VALUES (IdNum, 'Martha Gray',
          'Beach scene, balls on water',
          ORDSYS.ORDVIR.init('FILE','ORDVIRDIR','virdemo2.dat'));

   SELECT photo INTO image FROM stockphotos WHERE photo_id = IdNum FOR UPDATE;

   -- Import calls setProperties by default.
   -- Read the data into the database and set properties.
      image.import(ctx);

   -- Generate the image signature.
      image.Analyze;

   -- Update the photo column with the contents of image.
   -- This also stores the signature and other image-specific attributes.
   UPDATE stockphotos SET photo = image WHERE photo_id = IdNum;
   COMMIT;
END;
/
```

3.5 Loading Images from External Files Using SQL*Loader

Oracle8*i* database utilities provide support for loading BLOBs using SQL*Loader. This utility can be used to load external image files into image tables in the database. For more information on SQL*Loader, see *Oracle8i Utilities*.

Example 3–5 is an example of a control file (load1.ctl) used to load images into a table of ORDVir objects. This example assumes that the two images are located on

drive e:\ordvirdir\ and that record 4 in the BEGINDATA section ends with a carriage return.

Example 3–5 Load Images Using SQL*Loader

```
LOAD DATA
INFILE *
INTO TABLE stockphotos
APPEND
FIELDS TERMINATED BY ',' 
(
  photo_id, photographer, annotation,
  photo column object
     (image column object
        (source column object
           (localData_fname FILLER CHAR(4000),
            localData LOBFILE(photo.image.source.localData_fname)
                 TERMINATED BY EOF
           )
        )
     )
 )
BEGINDATA
3, "John Doe", "vir image 3", e:\ordvirdir\virdemo1.dat,
4, "Jane Doe", "vir image 4", e:\ordvirdir\virdemo2.dat,

E:\Scripts\ORDVir>sqlldr userid=vir1/vir1 control=load1.ctl log=load1.log

SQL*Loader: Release 8.1.5.0.0 - Production on Thu May 25 10:14:02 2000

(c) Copyright 1999 Oracle Corporation.  All rights reserved.

Commit point reached - logical record count 2
```

SQL*Loader does not set the properties of the images for you automatically. If you know the properties of the images ahead of time, you can specify the respective column names in the control file and the respective properties in each data row. In addition, you must also invoke the analyze method to generate an image signature for each image loaded and update the row. Example 3–6 shows a PL/SQL script that does both of these tasks.

Example 3–6 Set Properties and Generate Signatures for the Loaded Images

```
DECLARE
   image     ORDSYS.ORDVIR;
```

```
BEGIN
 FOR I IN 3..4 LOOP

   SELECT photo INTO image FROM stockphotos WHERE photo_id = I FOR UPDATE;

   -- Set property attributes for the image data.
      image.SetProperties;

   -- Generate the image signature.
      image.Analyze;

   -- Update the photo column with the contents of image.
   -- This also stores the signature and other image-specific attributes.
   UPDATE stockphotos SET photo = image WHERE photo_id = I;
   COMMIT;
 END loop;
END;
/
```

3.6 Retrieving an Image

Example 3–7 reads an image from the table and prepares it to be passed along, either directly to the end user or to the application for further processing. The program segment selects the desired photograph (where `photo_id = myid`) and places it in an image storage area.

Example 3–7 Retrieve an Image (Simple Read)

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON

DECLARE
   image     ORDSYS.ORDVIR;
   myid      INTEGER :=1;
BEGIN
   -- Select the desired photograph from the stockphotos table.
   SELECT photo INTO image FROM stockphotos
          WHERE photo_id = myid;
END;
/
```

3.7 Retrieving Images Similar to a Comparison Image (Content-Based Retrieval)

Example 3–8 performs content-based retrieval; it finds images that are similar to an image chosen for comparison.

The program segment performs the following operations:

1. Defines a cursor to perform the matching. The cursor sets the following weight values:
   - Global color: 0.2
   - Local color: 0.3
   - Texture: 0.1
   - Structure: 0.4
2. Select photo with ID=1 as the comparison image.
3. Generates the signature (`compare_sig`) of the comparison image (`compare_img`). Note: The program must have previously placed the comparison image in `compare_img`.
4. Sets the threshold value at 25.
5. Selects the matching images, using the cursor.

Example 3–8 Retrieve Images Similar to a Comparison Image

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON

DECLARE
   threshold    NUMBER;
   compare_sig  RAW(2000);
   compare_img  ORDSYS.ORDVir;
   photo_id     NUMBER;
   photographer VARCHAR2(64);
   annotation   VARCHAR2(255);
   photo        ORDSYS.ORDVIR;

-- Define cursor for matching. Set weights for the visual attributes.
CURSOR getphotos IS
   SELECT photo_id, photographer, annotation, photo FROM stockphotos T
   WHERE ORDSYS.VIRSimilar(T.photo.Signature, compare_sig,
```

```
                'globalcolor="0.2" localcolor="0.3" texture="0.1"
                   structure="0.4"', threshold)=1 AND photo_id <> 1;

BEGIN
 -- Create BLOB object
   SELECT s.photo INTO compare_img FROM stockphotos s
     WHERE photo_id = 1;

 -- Generate signature of comparison image, which resides in compare_img.
   compare_img.Analyze;
   compare_sig:= compare_img.signature;

   -- Set the threshold value.
   threshold := 25;

   -- Retrieve rows for matching images.
   OPEN getphotos;
   LOOP
      FETCH getphotos INTO photo_id, photographer, annotation, photo;
      EXIT WHEN getphotos%NOTFOUND;
      -- Display or store the results.
      -- .
      -- .
   END LOOP;
   CLOSE getphotos;
END;
/
```

Example 3–9 finds the photo_id and score of the image that is most similar to a comparison image with respect to texture. None of the other image characteristics is considered. This example uses the VIRScore( ) operator, which is an ancillary operator used in conjunction with the VIRSimilar( ) operator. The parameter passed to VIRScore( ) (123 in this example) is a reference to the same parameter passed in the call to VIRSimilar( ). In this example, three of the three images compared to the comparison image were identical to the comparison image and showed a score of zero (0).

Example 3–9 Find photo_id and Score of Similar Image

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON

SELECT Q.photo_id,
```

```
       ORDSYS.VIRScore(123) SCORE
       FROM stockphotos Q, stockphotos S
       WHERE S.photo_id=1 AND Q.photo_id != S.photo_id AND
             ORDSYS.VIRSimilar(Q.photo.signature, S.photo.signature,
                               'texture=1', 20.0, 123)=1;
 PHOTO_ID     SCORE
--------- ---------
        2         0
        3         0
        4         0
```

3.8 Creating a Domain Index

To improve performance, you can create a domain index on the image signature attribute. Example 3–10 creates an index called imgindex.

Example 3–10 Create a VIR Index

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON

CREATE INDEX imgindex ON stockphotos(photo.signature)
 INDEXTYPE IS ordsys.ordviridx
 PARAMETERS('ORDVIR_DATA_TABLESPACE = tbs_1,ORDVIR_INDEX_TABLESPACE = tbs_2');
```

As with any index, the tablespace (tbs_1 and tbs_2) must be created first.

The following recommendations are good starting points for further index tuning:

- ORDVIR_DATA_TABLESPACE -- Each signature requires approximately 350 bytes in this tablespace. The tablespace should be at least 350 times the number of signatures in the table.
- ORDVIR_INDEX_TABLESPACE -- The size of the tablespace should be 100 times the size of the initial and final extents specified. For example, if an extent is 10 KB, the tablespace size should be 1 MB. The initial and final extents should be equal to each other. The size of the tablespace should also be approximately equal to the size of ORDVIR_DATA_TABLESPACE.
- Typically, it will be much faster if you create the index after the images are loaded into the database and analyzed.

  Note that if a signature is corrupt, or is in incorrect byte order for your system, the CREATE INDEX statement exits and index creation is not completed. This

incomplete index is not usable. Delete the index, delete the corrupt signature or correct the byte order (see the Convert( ) Operator), and re-create the index.

To prevent wasting time when creating an index for large tables, ensure that the signatures are in the correct format before creating the index. This can be accomplished by generating the signatures on the same system as they are used, or by using the Convert( ) operator.

- Creating an index for large tables can be very time consuming. When importing a large number of images, you should postpone index creation until after the import operation completes. Do this by specifying the following parameters to the IMPORT statement: INDEXES=N and INDEXNAME=<filename>. See *Oracle8i Utilities* for details.
- Rollback segments of an appropriate size are required. The size depends on the size of your transactions, such as, how many signatures are indexed at one time.
- Analyze the new index. See Section 2.4.

3.9 Retrieving Images Similar to a Comparison Image Using Index Operations (Indexed Content-Based Retrieval)

Queries for indexed and nonindexed comparisons are identical. The Oracle optimizer uses the domain index if it determines that the first argument passed to the VIRSimilar( ) operator is a domain-indexed column. Otherwise, the optimizer invokes a functional implementation of the operator that compares the query signature with the stored signatures, one row at a time.

See Section 3.7 for examples of retrieving similar images. As in the example, be sure to specify the query signature as the second parameter.

3.10 Converting an Image from One Format to Another Format

Example 3–11 converts an image from its current format to GIF format for display on a Web page. The program segment performs the following operations:

1. Selects the desired photograph (where `photo_id = 1`) and places it in an image storage area.
2. Uses the process( ) method to convert the format to GIFF. (You do not need to know the current image format.)
3. Updates the `photo` column with content of the converted image.

Example 3–11 Convert an Image to a Different Format

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON

DECLARE
   image     ORDSYS.ORDVIR;

BEGIN

   -- Select the desired photograph from the stockphotos table.
   SELECT photo INTO image FROM stockphotos WHERE photo_id = 1 FOR UPDATE;

   -- Use Process method to perform the conversion.
   image.Process('fileFormat=GIFF');
   -- Update the photo column with the contents of image.
   UPDATE stockphotos SET photo = image WHERE photo_id = 1;
   COMMIT;

END;
/
```

3.11 Extending the Object Type

You can use the ORDVir type as the basis for a new type of your own creation.

For example, the original table created in Section 3.2 had a column called *annotation*. You could move that annotation into a new object encapsulating the ORDVir type. This new type can have additional attributes and methods.

To simulate subtyping (which is not supported in this release), you can define wrappers for the ORDVir methods and access functions. Example 3–12 adds the annotation column to the ORDVir type and defines wrappers for three procedures, making a new type called AnnotatedImage. The ellipses in this example indicate that more methods could be included, but they are not being shown.

Example 3–12 Extend the ORDVir Type

```
CREATE TYPE AnnotatedImage AS OBJECT
   (   image      ORDSYS.ORDVIR,
       annotation VARCHAR2(2000),

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

```
                                    dest IN OUT AnnotatedImage)
-- CONSTRUCTORS
       STATIC FUNCTION init RETURN AnnotatedImage,
       STATIC FUNCTION init(srcType IN VARCHAR2,
                            srcLocation IN VARCHAR2,
                            srcName IN VARCHAR2,
                            annotation IN VARCHAR2) RETURN AnnotatedImage
       --.
       --.
       --.
    );
/

CREATE TYPE BODY AnnotatedImage AS
  MEMBER PROCEDURE SetProperties(SELF IN OUT AnnotatedImage) IS
  BEGIN
    SELF.image.setProperties;
  END SetProperties;

  MEMBER PROCEDURE Copy(dest IN OUT AnnotatedImage) IS
  BEGIN
    SELF.image.copy(dest.image);
    dest.annotation := SELF.annotation;
  END Copy;

  MEMBER PROCEDURE ProcessCopy(command in VARCHAR2, dest IN OUT AnnotatedImage)
   IS
  BEGIN
    SELF.image.processCopy(command,dest.image);
    dest.annotation := SELF.annotation;
  END ProcessCopy;

-- CONSTRUCTORS
  STATIC FUNCTION init RETURN AnnotatedImage IS
    tmpImage ORDSYS.ORDVir;
  BEGIN
    tmpImage := ORDSYS.ORDVir.init;
    RETURN AnnotatedImage(
              tmpImage,              -- ORDVir
              NULL);                 -- annotation
  END init;

  STATIC FUNCTION init( srcType IN VARCHAR2,
                        srcLocation IN VARCHAR2,
                        srcName in VARCHAR2,
```

```
                              annotation IN VARCHAR2) RETURN AnnotatedImage IS
    tmpImage ORDSYS.ORDVir;
  BEGIN
    tmpImage := ORDSYS.ORDVir.init(srcType,srcLocation,srcName);
    RETURN AnnotatedImage(
              tmpImage,               -- ORDVir
              annotation);            -- annotation
  END init;
 -- .
 -- .
 -- .
 END;
/
SHOW ERRORS
```

After creating the new type, you can use it as you would any other type. Notice the static constructor function, init( ), in Example 3–13: newType(ORDSYS.ORD-Vir.init(). Because user-defined constructors are not supported in this release, inserting into a simulated subtype is only possible by being aware of the full encapsulation hierarchy or using the ORDVir.init( ) method.

Example 3–13 Use an Extended Type

```
CONNECT vir1/vir1;
SET SERVEROUTPUT ON
SET ECHO ON

CREATE TABLE table my_example (id NUMBER,an_image AnnotatedImage);

INSERT INTO my_example VALUES ( 1,
    AnnotatedImage.init('FILE','ORDVIRDIR','virdemo1.dat'),
    'some text describing the image'));
COMMIT;

DECLARE
  myimage AnnotatedImage;
BEGIN
  SELECT an_image INTO myimage FROM my_example FOR UPDATE;

  myimage.setProperties;

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

  UPDATE my_example SET an_image=myimage;
```

```
END;
/
This image has a description of
This is an example of using the VIR object as a subtype

PL/SQL procedure successfully completed.
```

3.12 Using Object Views

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

Oracle Corporation 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 does not have attributes containing sensitive data and does not 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.

In Example 3–14, consider the following relational table (containing no ORDVir objects).

Example 3–14 Show the Relational Table Containing No ORDVir Object

```
CREATE TABLE flat (
   id                NUMBER,
   localData         BLOB,
   srcType           VARCHAR2(4000),
   srcLocation       VARCHAR2(4000),
   srcName           VARCHAR2(4000),
   updateTime        DATE,
   local             NUMBER,
   height            INTEGER,
   width             INTEGER,
   contentLength     INTEGER,
   fileFormat        VARCHAR2(4000),
   contentFormat     VARCHAR2(4000),
   compressionFormat VARCHAR2(4000),
   mimeType          VARCHAR2(4000),
   signature         RAW(2000));
```

You can create an object view on the relational table shown in Example 3–14 as follows in Example 3–15.

Example 3–15 Show the Object View Containing the ORDVir Object and Relational Columns

```
CREATE OR REPLACE VIEW object_images_v AS
  SELECT
      id,
      ORDSYS.ORDVir(
        ORDSYS.ORDImage(
          ORDSYS.ORDSource(
             T.localData, T.srcType, T.srcLocation, T.srcName,
              T.updateTime, T.local),
          T.height,
          T.width,
          T.contentLength,
          T.fileFormat,
          T.contentFormat,
          T.compressionFormat,
          T.mimeType),
        T.signature) IMAGE
   FROM flat T;
```

Object views provide the flexibility of looking at the same relational or object data in more than one way. You can use different in-memory object representations for different applications without changing the way you store the data in the database. Object views also provide a way to use replication when your application uses objects. You can create an object view containing one or more object columns and also use replication. See *Oracle8i Concepts* for more information on defining, using, and updating object views.

3.12.1 Scripts for Creating and Populating a Visual Information Retrieval Image Table from a BFILE Data Source

The following scripts create and load a Visual Information Retrieval image table from a BFILE data source.

The following set of scripts:

1. Creates a tablespace for the image data, creates a user and grants certain privileges to this new user, creates an image data load directory (create_viruser.sql).

2. Creates a table with two columns, inserts two rows into the table and initializes the object column to empty with a locator (create_virtable.sql).
3. Loads the image data with a SELECT FOR UPDATE operation using an import method to import the data from a BFILE (importvirimg.sql).
4. Performs a check of the properties for the loaded data to ensure that it is really there (chkprop.sql).

The fifth script (setup_virschema.sql) automates this entire process by running each script in the required order. The last script (readvimage.sql) creates a stored procedure that performs a SELECT operation to read a specified amount of image data from the BLOB beginning at a particular offset until all the image data is read. To successfully load the image data, you must have a *virdir* directory created on your system containing the virdemo1.dat and virdemo2.dat files, and this directory and disk drive must be specified in the CREATE DIRECTORY statement in the create_viruser.sql file.

Script 1: Create a Tablespace, Create an VIR Image User, Grant Privileges to the VIR Image User, and Create an VIR Image Data Load Directory (create_viruser.sql)

This script creates the virdemo tablespace with a data file named virdemo.dbf of 200 MB in size, with an initial extent of 64 K, a next extent of 128 K, and turns on table logging. Next, the viruser user is created and given connect, resource, create library, and create directory privileges followed by creating the image data load directory.

Note: You must edit the create_viruser.sql file and either enter the system password in the connect statement or comment out the connect statement, and run this file in the system account. You must specify the disk drive in the CREATE DIRECTORY statement. Also, create the temp temporary tablespace if you have not already created it, otherwise this file will not run.

```
-- create_viruser.sql

-- Connect as admin
connect system/<system password>;

-- Edit this script and either enter your system password here
-- to replace <system password> or comment out this connect
-- statement and connect as system before running this script.
```

```
set serveroutput on;
set echo on;

-- Need system manager privileges to delete a user.
-- Note: There is no need to delete viruser user, if you do not delete
-- the virdemo tablespace, therefore comment out next line.

-- drop user viruser cascade;

-- Need system manager privileges to delete a directory. If there is no need
-- to really delete it, then comment out next line.

-- drop directory virdir;

-- Delete then create tablespace.

-- Note: It is better to not delete and create tablespaces,
-- so comment this next line out. The create tablespace statement
-- will fail because it already exists.

-- drop tablespace virdemo including contents;

-- If you do not make the preceding a comment and really want to delete the
-- virdemo tablespace, remember to manually delete the virdemo.dbf
-- file to complete the operation. Otherwise, you cannot create
-- the virdemo tablespace again because the virdemo.dbf file already
-- exists. Therefore, it might be best to create this tablespace
-- once and not delete it.

-- Create tablespace.
create tablespace virdemo
       datafile 'virdemo.dbf' size 200M
       minimum extent 64K
       default storage (initial 64K next 128K)
       logging;

-- Create viruser user.
create user viruser identified by viruser
default tablespace virdemo
temporary tablespace temp;
-- Note: If you do not have a temp tablespace already defined, you will
-- have to create it first for this script to work.
```

```
grant connect, resource, create library to viruser;
grant create any directory to viruser;

-- Note: If this user already exists, you get an error message
-- when you try to create this user again.

-- Connect as viruser.
connect viruser/viruser

-- Create the virdir load directory; this is the directory where
-- the image files are residing.
create or replace directory virdir
       as 'e:\oracle\ord\vir\demo';
grant read on directory virdir to public with grant option;

-- Note: If this directory already exists, an error message will
-- be returned stating so, and the operation will fail;
-- ignore the message.
```

Script 2: Create the VIR Image Table and Initialize the Column Object (create_virtable.sql)

This script creates the VIR image table and then performs an insert operation to initialize the column object to empty for two rows. Initializing the column object creates the BLOB locator that is required for loading each row with BLOB data in a subsequent data load operation.

```
-- create_virtable.sql

connect viruser/viruser;
set serveroutput on;
set echo on;

drop table virtable;
create table virtable (id number,
       virImage ordsys.ordvir);

-- Insert a row with empty BLOB.
insert into virtable values(1,ORDSYS.ORDVIR.init());

-- Insert a row with empty BLOB.
insert into virtable values(2,ORDSYS.ORDVIR.init());
commit;
```

Script 3: Load the Image Data (importvirimg.sql)

This script performs a SELECT FOR UPDATE operation to load the VIR image data by first setting the source for loading the VIR image data from a file, importing the data, setting the properties for the BLOB data, updating the row, and committing the transaction. To successfully run this script, you must copy two VIR image files to your VIRDIR directory using the names specified in this script, or modify this script to match the file names of your image files, point VIRDIR to the Visual Information Retrieval demo directory.

```
-- importvirimg.sql

set serveroutput on
set echo on

-- Import the two files into the database.

DECLARE
  obj ORDSYS.ORDVIR;
  ctx RAW(4000) := NULL;

BEGIN
-- This imports the image file virdemo1.dat from the VIRDIR directory
-- on a local file system (srcType=FILE) and sets the properties.

  select virImage into obj from virtable where id = 1 for update;
  obj.setSource('FILE','VIRDIR','virdemo1.dat');
  obj.import(ctx);

  update virtable set virimage = obj where id = 1;
  commit;

-- This imports the image file virdemo2.dat from the VIRDIR directory
-- on a local file system (srcType=FILE) and sets the properties.

  select virImage into obj from virtable where id = 2 for update;
  obj.setSource('FILE','VIRDIR','virdemo2.dat');
  obj.import(ctx);

  update virtable set virimage = obj where id = 2;
  commit;
END;
/
```

Script 4: Check the Properties of the Loaded Data (chkprop.sql)

This script performs a SELECT operation of the rows of the VIR image table, then gets the image characteristics of the BLOB data to check that the BLOB data is in fact loaded.

```
-- chkprop.sql

set serveroutput on;

--Connect viruser/viruser.
--Query virtable for ORDSYS.ORDVir.

DECLARE
  vimage ORDSYS.ORDVir;
  idnum integer;
  properties_match BOOLEAN;

BEGIN
 FOR I IN 1..2 LOOP
  SELECT id into idnum from virtable where id=I;
   dbms_output.put_line('image id:          '|| idnum);

  SELECT virImage into vimage from virtable where id=I for update;

  properties_match := vimage.checkProperties;
  IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
  END IF;

   dbms_output.put_line('image height:      '|| vimage.getHeight);
   dbms_output.put_line('image width:       '|| vimage.getWidth);
   dbms_output.put_line('image MIME type:   '|| vimage.getMimeType);
   dbms_output.put_line('image file format: '|| vimage.getFileFormat);
   dbms_output.put_line('BLOB Length:       '||
TO_CHAR(vimage.getContentLength));
   dbms_output.put_line('-----------------------------------------');

 END loop;
END;
/
```

The folllowing are the results from running the script chkprop.sql:

```
SQL> @chkprop.sql
image id:          1
Check Properties Succeeded
```

```
image height:      400
image width:       600
image MIME type:   image/bmp
image file format: BMPF
BLOB Length:       720054
-----------------------------------------
image id:          2
Check Properties Succeeded
image height:      400
image width:       600
image MIME type:   image/bmp
image file format: BMPF
BLOB Length:       720054
-----------------------------------------

PL/SQL procedure successfully completed.
```

Automated Script (setup_virschema.sql)

This script runs each of the previous four scripts in the correct order to automate this entire process.

```
-- setup_virschema.sql
-- Create viruser user, tablespace, and load directory to
-- hold the files.
@create_viruser.sql

-- Create VIR imame table.
@create_virtable.sql

--Import 2 images and set properties.
@importvirimg.sql

--Check the properties of the images.
@chkprop.sql

--exit;
```

Read Data from the BLOB (readvimage.sql)

This script performs a SELECT operation to read a specified amount of image data from the BLOB, beginning at a particular offset until all the image data is read.

```
-- readvimage.sql

set serveroutput on;
```

```
set echo on;

create or replace procedure readvimage as

-- Note: ORDVir has no readFromSource method like ORDAudio
-- and ORDVideo; therefore, you must use the DBMS_LOB package to
-- read image data from a BLOB.

   buffer RAW (32767);
   src BLOB;
   obj ORDSYS.ORDVir;
   amt BINARY_INTEGER := 32767;
   pos integer := 1;
   read_cnt integer := 1;

BEGIN

   Select  t.virimage.getcontent into src from virtable t where t.id = 1;
   Select virimage into obj from virtable t where t.id = 1;

    DBMS_OUTPUT.PUT_LINE('Content length is: '|| TO_CHAR(obj.getContentLength));

   LOOP
         DBMS_LOB.READ(src,amt,pos,buffer);
           DBMS_OUTPUT.PUT_LINE('start position: '|| pos);
           DBMS_OUTPUT.PUT_LINE('doing read  '|| read_cnt);
         pos := pos + amt;
         read_cnt := read_cnt + 1;

-- Note: Add your own code here to process the image data being read;
-- this routine just reads data into the buffer 32767 bytes
-- at a time, then reads the next chunk, overwriting the first
-- buffer full of data.

   END LOOP;

EXCEPTION

   WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('----------------');
   DBMS_OUTPUT.PUT_LINE('End of data ');

END;

/
```

```
show errors
```

To execute the stored procedure, enter the following SQL statements:

```
SQL> SET SERVEROUTPUT ON;
SQL> EXECUTE readvimage;
Content length is: 720054
start position: 1
doing read  1
start position: 32768
doing read  2
start position: 65535
doing read  3
.
.
.
start position: 688108
doing read  22
---------------
End of data

PL/SQL procedure successfully completed.
```

4

Visual Information Retrieval Reference

The Visual Information Retrieval library consists of:

- Ensuring future compatibility with an evolving ORDVir object type -- See Section 4.1
- Object type -- See Section 4.2.
- Constructors -- See Section 4.3.
- Methods -- See Section 4.4.
- Operators -- See Section 4.5.

The examples in this chapter assume that a table called stockphotos has been created and filled with some photographic images. The table was created using the following SQL statement:

```
CREATE TABLE stockphotos (photo_id NUMBER, photographer VARCHAR2(64),
            annotation VARCHAR2(255), photo ORDSYS.ORDVIR);
```

When you are storing or copying images, you must first create an empty BLOB in the table. The following method invocation creates an empty Visual Information Retrieval object:

```
ORDSYS.ORDVIR.init();
```

Methods invoked at the ORDSource level that are handed off to the source plug-in for processing have ctx(RAW(4000)) as their first argument. Before calling any of these methods for the first time, the client must perform the following tasks:

1. Allocate the ctx structure.
2. Initialize the ctx structure to null.
3. Invoke the source.open( ) method.

At this point, the source plug-in can initialize the context for this client.

4. When processing is complete, the client should invoke the source.close( ) method.

See *Oracle interMedia Audio, Image, and Video User's Guide and Reference* for descriptions of source methods.

Note: If you manipulate the image data itself by either directly modifying the BLOB, or changing the external source, then you must ensure that the object attributes stay synchronized and the update time is modified. Otherwise, the object attributes will not match the image data.

4.1 Ensuring Future Compatibility with an Evolving ORDVir Object Type

Oracle Corporation may evolve the ORDVir object type by adding new object attributes in a future release of Visual Information Retrieval. Client-side applications that you want to maintain compatibility to the 8.1.7 release of the ORDVir object type even after a server upgrade that evolves the object type are advised to do the following:

- Make a call to the compatibility initialization function at the beginning of the application, if necessary (see Section 4.1.1).
- Use the static constructor functions, init( ), in INSERT statements that are provided beginning with release 8.1.7 (see Section 4.3). Do not use the default constructors because INSERT statements using the default constructors will fail if the *inter*Media object types have evolved to add new attributes.

Note: If you do not do the preceding recommended actions, you may have to upgrade and perhaps even recompile your application when you upgrade to a newer server version that evolves the type.

4.1.1 When and How to Call the Compatibility Initialization Function

Only client-side applications that statically understand the structure of the ORDVir object type need to make a call to the compatibility initialization function. Server-side stored procedures automatically use the newly installed (potentially evolved) ORDVir object type after an upgrade so you do not need to call the compatibility initialization function from server-side stored procedures.

Client-side applications that do not statically (at compile time) understand the structure of ORDVir object type do not need to call the compatibility initialization function. Oracle Call Interface (OCI) applications that determine the structure of the ORDVir object type at runtime through the OCIDescribeAny call do not need to call the compatibility initialization function.

Client-side applications written in OCI that have been compiled with the C structure of an ORDVir object type (generated by the Object Type Translator (OTT)) should make a call to the server-side PL/SQL function, ORDSYS.VIR.compatibilityInit( ), at the beginning of the application. See the reference description of this function later in this section.

Client-side applications written in Java using the Visual Information Retrieval Java Classes for 8.1.7 should call the OrdMediaUtil.virCompatibilityInit( ) function after connecting to an Oracle database.

```
public static void virCompatibilityInit(OracleConnection con)
    throws Exception
```

This Java function takes OracleConnection as an argument. The included Visual Information Retrieval 8.1.7 Java API ensures that your 8.1.7 application will work (without upgrading) with a potential future version of Visual Information Retrieval with an evolved object type.

There is not yet a way for client-side PL/SQL applications to maintain compatibility to the 8.1.7 release of the ORDVir object type if the objects are evolved to add new attributes in a future release.

See the reference description of the compatibilityInit( ) function later in this section and the *Oracle Visual Information Retrieval Java Classes User's Guide and Reference*, release 8.1.7 for further information along with detailed descriptions and examples. This guide will be available on the Oracle Technology Network at

`http://technet.oracle.com/`

compatibilityInit( ) Method

Format

```
compatibilityInit(release IN    VARCHAR2,
                  errmsg OUT VARCHAR2)
RETURN NUMBER;
```

Description

Allows for compatibly evolving the ORDVir object type in a future release.

Parameters

release
The string input parameter. This string should be set to '8.1.7' to allow an 8.1.7 application to work (without upgrading) with a potential future release of the Oracle database server and Visual Information Retrieval with an evolved ORDVir object type.

errmsg
The string output parameter. If this function returns a status other than 0, this errmsg string contains the reason for the failure.

Pragmas

None.

Exceptions

None.

Usage Notes

You should begin using the compatibilityInit( ) method as soon as possible to ensure that you will not have to upgrade the Oracle software on your client node or recompile your client application to work with a future release of the Oracle database server if the ORDVir object type evolves in a future release. See Section 4.1.1 to determine if you need to call this function.

The compatibility initialization function for Visual Information Retrieval is located in the ORDSYS.VIR package.

Examples

This example uses OCI and sets the compatililityInit( ) method release parameter to release 8.1.7 to allow an 8.1.7 application to work with a future release of Oracle and ORDVir with an evolved object type; note, that this is not a standalone program in that it assumes that you have allocated handles beforehand:

```
void prepareExecuteStmt( OCIEnv *envHndl,
            OCIStmt **stmtHndl,
            OCIError *errorHndl,
            OCISvcCtx *serviceCtx,
            OCIBind *bindhp[] )
{
   text *statement = (text *)
      "begin :sts := ORDSYS.VIR.compatibilityInit( :vers, :errText );
end;";
  sword sts = 0;
  text *vers = (text *)"8.1.7";
  text errText[512];
  sb2 nullInd;

  printf( " Preparing statement\n" );

  OCIHandleAlloc( envHndl, (void **) stmtHndl, OCI_HTYPE_STMT, 0, NULL
);

  OCIStmtPrepare( *stmtHndl, errorHndl, (text *)statement,
        (ub4)strlen( (char *)statement ), OCI_NTV_SYNTAX,
        OCI_DEFAULT );

  printf( " Executing statement\n" );

  OCIBindByPos( *stmtHndl, &bindhp[ 0 ], errorHndl, 1, (void *)&sts,
       sizeof( sts ), SQLT_INT, (void *)0, NULL, 0, 0,
       NULL, OCI_DEFAULT );

  OCIBindByPos( *stmtHndl, &bindhp[ 1 ], errorHndl, 2, vers,
       strlen((char *)vers) + 1, SQLT_STR, (void *)0, NULL,
       0, 0, NULL, OCI_DEFAULT );

  OCIBindByPos( *stmtHndl, &bindhp[ 2 ], errorHndl, 3, errText,
       sizeof( errText ), SQLT_STR, &nullInd, NULL, 0, 0,
```

```
        NULL, OCI_DEFAULT);

  OCIStmtExecute( serviceCtx, *stmtHndl, errorHndl, 1, 0,
        (OCISnapshot *)NULL, (OCISnapshot *)NULL, OCI_DEFAULT );

  printf( " Statement executed\n" );
  if (sts != 0)
    {
    printf( "CompatibilityInit failed with Sts = %d\n", sts );
    printf( "%s\n", errText );
    }


}
```

4.2 Object Type

The Visual Information Retrieval object type ORDVir is composed of the ORDImage type and a signature attribute. The ORDImage type is composed of the ORDSource type and a number of additional descriptive attributes.

This section presents reference information on the object types.

For more information on BLOBs and BFILEs, see *Oracle8i Application Developer's Guide - Large Objects (LOBs).* For more information on the ORDImage and ORDSource types, see *Oracle interMedia Audio, Image, and Video User's Guide and Reference*.

ORDVir Object Type

The ORDVir object type supports storage and retrieval of image data stored in a BLOB, external file, or referenced by a URL. This object type is defined as follows:

```
CREATE TYPE ORDVIR AS OBJECT
(
-- TYPE ATTRIBUTES
image               ORDImage,
signature           RAW(2000)
-- METHOD DECLARATION
-- CONSTRUCTORS
STATIC FUNCTION init( ) RETURN ORDVir,
STATIC FUNCTION init(srcType     IN VARCHAR2,
                     srcLocation IN VARCHAR2,
                     srcName     IN VARCHAR2) RETURN ORDVir,

 -- IMAGE COPY METHOD
  MEMBER PROCEDURE copy(dest IN OUT NOCOPY ORDVir),

 -- IMAGE PROCESSING RELATED METHODS
  MEMBER PROCEDURE process(SELF    IN OUT ORDVir,
                           command IN     VARCHAR2),
  MEMBER PROCEDURE processCopy(command IN     VARCHAR2,
                               dest    IN OUT NOCOPY ORDVir),

 -- IMAGE PROPERTY SET AND CHECK METHODS
  MEMBER PROCEDURE setProperties(SELF IN OUT ORDVir),
  MEMBER PROCEDURE setProperties(SELF IN OUT ORDVir,
                                 description IN VARCHAR2),
  MEMBER FUNCTION checkProperties RETURN BOOLEAN,

   -- IMAGE ATTRIBUTE ACCESSOR METHODS
  MEMBER FUNCTION getHeight RETURN INTEGER,
  PRAGMA RESTRICT_REFERENCES(getHeight, WNDS, WNPS, RNDS, RNPS),

  MEMBER FUNCTION getWidth RETURN INTEGER,
  PRAGMA RESTRICT_REFERENCES(getWidth, WNDS, WNPS, RNDS, RNPS),

  MEMBER FUNCTION getFileFormat RETURN VARCHAR2,
  PRAGMA RESTRICT_REFERENCES(getFileFormat, WNDS, WNPS, RNDS, RNPS),

  MEMBER FUNCTION getContentFormat RETURN VARCHAR2,
```

```
PRAGMA RESTRICT_REFERENCES(getContentFormat, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getCompressionFormat RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getCompressionFormat, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION getSignature RETURN RAW,
PRAGMA RESTRICT_REFERENCES(getSignature, WNDS, WNPS, RNDS, RNPS),
-- DATE RELATED METHODS
MEMBER FUNCTION  getUpdateTime RETURN DATE,
PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setUpdateTime(current_time DATE),

-- MIMETYPE RELATED METHODS
MEMBER FUNCTION getMimeType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getMimeType, WNDS, WNPS, RNDS, RNPS),
MEMBER PROCEDURE setMimeType(mimetype IN VARCHAR),

-- SOURCE/CONTENT RELATED METHODS
MEMBER FUNCTION  getContentLength RETURN INTEGER,
PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION  getContent RETURN BLOB,
PRAGMA RESTRICT_REFERENCES(getContent, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION  getBFILE RETURN BFILE,
PRAGMA RESTRICT_REFERENCES(getBFILE, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE deleteContent,

MEMBER PROCEDURE setSource(source_type     IN VARCHAR2,
                           source_location IN VARCHAR2,
                           source_name     IN VARCHAR2),
MEMBER FUNCTION  getSource RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSource, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION  getSourceType RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION  getSourceLocation RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS),

MEMBER FUNCTION  getSourceName RETURN VARCHAR2,
PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS),

MEMBER PROCEDURE import(ctx IN OUT RAW),
```

```
  MEMBER PROCEDURE importFrom(ctx IN OUT RAW,
                          source_type IN VARCHAR2,
                          source_location IN VARCHAR2,
                          source_name IN VARCHAR2),
  MEMBER PROCEDURE export(ctx      IN OUT RAW,
                          source_type     IN VARCHAR2,
                          source_location IN VARCHAR2,
                          source_name     IN VARCHAR2),

  MEMBER PROCEDURE setLocal,
  MEMBER PROCEDURE clearLocal,
  MEMBER FUNCTION isLocal RETURN BOOLEAN,
  PRAGMA RESTRICT_REFERENCES(isLocal, WNDS, WNPS, RNDS, RNPS),

  -- MIGRATION RELATED METHODS
  MEMBER PROCEDURE migrateFromORDVirB(
                        SELF IN OUT ORDVIR,
                        old_object IN ORDVirB),

  MEMBER PROCEDURE migrateFromORDVirF(
                        SELF IN OUT ORDVIR,
                        old_object IN ORDVirF),

  -- VIRAGE RELATED METHODS
  MEMBER PROCEDURE analyze(SELF IN OUT ORDVir)
);
```

The ORDImage object type consists of the fields shown in Table 4–1.

Table 4–1 ORDImage Object Type

Attribute	Data Type	Description
source	ORDSource	Points to the source of the stored image data
height	INTEGER	Height of the image in pixels
width	INTEGER	Width of the image in pixels
contentLength	INTEGER	Size of the on-disk image file in bytes
fileFormat	VARCHAR2(4000)	File type of image (such as, TIFF or JFIF)
contentFormat	VARCHAR2(4000)	Type of image (such as, monochrome or 8-bit grayscale

Table 4–1 ORDImage Object Type (Cont.)

Attribute	Data Type	Description
compressionFormat	VARCHAR2(4000)	Compression type of the image
mimeType	VARCHAR2(4000)	MIME type of the image

The ORDSource object type consists of the fields shown in Table 4–2.

Table 4–2 ORDSource Object Type

Attribute	Data Type	Description
localData	BLOB	Locally stored image data.
srcType	VARCHAR2(4000)	Identifies the source type as one of the following: ■ NULL ■ 'FILE' -- source is an external file ■ 'HTTP' -- HTTP server name ■ '<name>' -- name of another source
srcLocation	VARCHAR2(4000)	Identifies where the data can be found based on the srcType. See Table 4–3.
srcName	VARCHAR2(4000)	Identifies the data object name based on the source type. See Table 4–4.
updateTime	DATE	The time at which the data was last updated.
local	NUMBER	Flag to determine if the data is local (1) or not (0).

Table 4–3 describes the valid srcLocation values.

Table 4–3 srcLocation Values

srcType	srcLocation
NULL	NULL or not accessed
FILE	<DIR> or name of the directory object

Table 4–3 srcLocation Values (Cont.)

srcType	srcLocation
HTTP	<SourceBase> or URL needed to find the base directory
<name>	<iden> or identifier string required to access another server

Table 4–4 describes the valid srcName values.

Table 4–4 Valid srcName Values

srcType	srcName
NULL	NULL or not accessed
FILE	<FILE> or name of the file
HTTP	<Source> or name of the object
<name>	<object name> or name of the object

In PL/SQL, data is moved with the DBMS LOB package. From the client, data is moved using OCI LOB calls. The ORDVir object type does not supply routines for moving data piece by piece.

4.3 Constructors

This section describes the constructor functions.

The Visual Information Retrieval constructor functions are as follows:

- init( )
- init(srcType, srcLocation,srcName)

init( ) Method

Format

init( ) RETURN ORDVir;

Description

Allows for easy initialization of instances of the ORDVir object type.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDVir attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE.
- source.local is set to 1 (local).
- source.localData is set to empty_blob.

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDVir object type, especially if the ORDVir type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor, (which initializes each object attribute) will fail under these circumstances.

Examples

Initialize the ORDVir object attributes:

```
DECLARE
  myImage ORDSYS.ORDVir;
```

```
BEGIN
  myImage := ORDSYS.ORDVir.init( );
INSERT INTO stockphotos VALUES (myImage);
END;
/
```

init( ) Method

Format

```
init(srcType      IN VARCHAR2,
     srcLocation IN VARCHAR2,
     srcName      IN VARCHAR2)
 RETURN ORDVir;
```

Description

Allows for easy initialization of instances of the ORDVir object type.

Parameters

srcType
The source type of the image data.

srcLocation
The source location of the image data.

srcName
The source name of the image data.

Pragmas

None.

Exceptions

None.

Usage Notes

This static method initializes all the ORDVir attributes to NULL with the following exceptions:

- source.updateTime is set to SYSDATE.
- source.local is set to 0.
- source.localData is set to empty_blob.

- source.srcType is set to the input value.
- source.srcLocation is set to the input value.
- source.srcName is set to the input value.

You should begin using the init( ) method as soon as possible to allow you to more easily initialize the ORDVir object type, especially if the ORDVir type evolves and attributes are added in a future release. INSERT statements left unchanged using the default constructor, (which initializes each object attribute) will fail under these circumstances.

Examples

Initialize the ORDVir object attributes:

```
DECLARE
  myImage ORDSYS.ORDVir;
BEGIN
  myImage := ORDSYS.ORDVir.init('FILE','VIRDIR','image1.gif');
INSERT INTO stockphotos VALUES (myImage);
END;
/
```

4.4 Methods for Image Manipulation

This section presents reference information on the methods used for image manipulation.

The Visual Information Retrieval methods are as follows:

- analyze( ): creates the signature of an image based on the image characteristics.
- checkProperties( ): verifies that the characteristics stored within an image object match the object attributes.
- clearLocal( ): resets the flag to indicate that data is stored externally.
- copy( ): copies an ORDVir image to another ORDVir object.
- deleteContent: deletes the contents of an image stored in a BLOB.
- export( ): transfers data from the local source to the specified external data source. Note that the only natively supported srcType that implements export is the "FILE" source. The HTTP source is not writable, therefore it does not support the export( ) method.
- getBFILE( ): returns the external image as a BFILE.
- getCompressionFormat( ): returns the compression type used on an image.
- getContent( ): returns the content of the local data.
- getContentFormat( ): returns the content type of the image.
- getContentLength( ): returns the size of the image in bytes.
- getFileFormat( ): returns the file type of an image.
- getHeight( ): returns the height of the image in pixels.
- getMimeType( ): returns the MIME type of an image.
- getSignature( ): returns the signature of an image.
- getSource( ): returns a string containing complete information about the external data source, formatted as a URL.
- getSourceLocation( ): returns the external source location of the image data.
- getSourceName( ): returns the external source name of the image data.
- getSourceType( ): returns the external source type of the image data.
- getUpdateTime( ): returns the time when the image object was last updated.

- getWidth( ): returns the width of an image in pixels.
- import( ): transfers data from an external data source to the local source.
- importFrom( ): transfers data from the specified external data source to the local source.
- isLocal( ): returns TRUE if the data is stored locally in a BLOB, or returns FALSE if the data is external.
- migrateFromORDVirB( ): copies an old ORDVirB image to an ORDVir object.
- migrateFromORDVirF( ): copies an old ORDVirF image to an ORDVir object.
- process( ): processes an image in place (for example, modifies it or converts it to another format); available for BLOBs only.
- processCopy( ): copies an image and processes the copy (for example, modifies it or converts it to another format); available for any source, but the destination must be a BLOB.
- setMimeType( ): sets the MIME type of the image. This method is called implicitly by any method that modifies natively supported images.
- setProperties( ): obtains and stores the attributes of an image for natively supported image formats. A second version of this method does the same for foreign images.
- setSource( ): sets the source information to where the external image data is located.
- setUpdateTime( ): sets the updateTime attribute for the image object, indicating when the image was last changed. This method is called implicitly by methods that modify natively supported images.

For more information on object types and methods, see *Oracle8i Concepts*.

Note that some of the following methods have **pragmas**, or compiler directives, associated with them. The pragma tells the PL/SQL compiler what sort of access the method needs to the database. The compiler can then deny the method read/write access to database tables, packaged variables, or both. Methods with defined pragma (such as the getXXX methods) can be called from SQL expressions.

Pragmas use the following syntax:

```
PRAGMA RESTRICT REFERENCES (function_name, WNDS, [,WNPS] [,RNDS] [RNPS]);
```

where:

- WNDS means writes no database state.

- WNPS means writes no package state.
- RNDS means reads no database state.
- RNPS means reads no package state.

analyze( ) Method

Format

analyze( );

Description

Analyzes an image, derives information relating to the visual attributes, and creates the image signature.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

The analyze( ) method creates the image attribute signature, which is necessary for any content-based retrieval. Whenever you are working with a new or changed image, you should also use the SetProperties( ) method to set the other image characteristics.

The analyze( ) method is functionally equivalent to the Analyze( ) operator. You must use the analyze( ) method, and not the operator, when working with foreign images (images not natively supported).

Examples

Create the signatures for all images in the stockphotos table.

```
DECLARE
   temp_image   ORDSYS.ORDVir;
   temp_id      INTEGER;
   cursor c1 is select id, image from stockphotos for update;
BEGIN
   OPEN c1;
```

```
   LOOP
      fetch c1 into temp_id, temp_image;
      EXIT WHEN c1%NOTFOUND;

 -- Generate signature and set the properties for the image.
      temp_image.analyze;
      temp_image.setProperties;
      UPDATE stockphotos SET photo = temp_image WHERE photo_id = temp_id;
   END LOOP;
   CLOSE c1;
END;
```

checkProperties( ) Method

Format

checkProperties RETURN BOOLEAN;

Description

Verifies that the properties stored in attributes of the image object match the properties of the image. This method should not be used for foreign images.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method to verify that the image attributes match the actual image.

Examples

Check the image attributes.

```
image            ORDSYS.ORDVir;
properties_match BOOLEAN;
BEGIN
  SELECT photo INTO image FROM stockphotos
    WHERE photo_id=1 FOR UPDATE;
  properties_match := image.checkProperties;
  IF properties_match THEN DBMS_OUTPUT.PUT_LINE('Check Properties Succeeded');
  END IF;
END;
```

clearLocal( ) Method

Format

clearLocal;

Description

Resets the local flag value from local (1) to nonlocal (0) for the source of the data.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

When the local flag is clear (0), this means that the data is stored externally. In that case, Visual Information Retrieval methods look for image data in the srcType, srcName, and srcLocation attributes.

Examples

Reset the local flag.

```
DECLARE
    Image ORDSYS.ORDVir;
BEGIN
    SELECT photo INTO Image FROM stockphotos WHERE photo_id = 1 FOR UPDATE;

    -- clear local so we look for image externally
    Image.clearLocal;

    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
END;
/
```

copy( ) Method

Format

copy(dest IN OUT NOCOPY ORDVir);

Description

Copies an image without changing it.

Parameters

dest
The destination of the new image.

Pragmas

None.

Exceptions

NULL_LOCAL_DATA -- Raised if the destination source.localData is not initialized, or if the source.isLocal value is 1 and source.localData is NULL.

Usage Notes

This method copies the image data, as is, including all source and image attributes, into the supplied local destination image.

If the data is stored locally in the source, then calling this method copies the BLOB to the destination source.localData field.

Calling this method copies the external source information to the external source information of the new image, whether or not the source data is stored locally.

Calling this method implicitly calls the setUpdateTime method on the destination object to update its timestamp information.

Examples

Create a copy of the image.

```
DECLARE
    Image_1 ORDSYS.ORDVir;
    Image_2 ORDSYS.ORDVir;
BEGIN
    SELECT photo,
       INTO Image_2
       FROM stockphotos
       WHERE photo_id = 1 FOR UPDATE;
    SELECT photo,
       INTO Image_1
       FROM stockphotos
       WHERE photo_id = 1;

    -- copy the data from Image_1 to Image_2
    Image_1.copy(Image_2);
    UPDATE stockphotos SET photo = Image_2
        WHERE photo_id = 2;
END;
```

deleteContent Method

Format

deleteContent;

Description

Deletes the local data from the current local source (localData).

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

This method can be called after you export the data from the local source to an external image data source and you no longer need this data in the local source.

Examples

Delete the local data from the current local source.

```
DECLARE
    Image ORDSYS.ORDVir;
BEGIN
    SELECT photo INTO Image FROM stockphotos
      WHERE photo_id = 1 FOR UPDATE;
     -- delete the local content of the image
    Image.deleteContent;
    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
END;
/
```

export( ) Method

Format

```
export(ctx              IN OUT RAW,
       source_type      IN VARCHAR2,
       source_location  IN VARCHAR2,
       source_name      IN VARCHAR2);
```

Description

Transfers image data from a local source (localData) within an Oracle database to an external image data source.

Parameters

ctx
The source plug-in context information.

source_type
The source type of the location where the image data is to be exported.

source_location
The location where the image data is to be exported.

source_name
The name of the image object where the image is to be exported.

Pragmas

None.

Exceptions

None.

Usage Notes

After exporting image data, all image attributes remain unchanged and srcType, srcLocation, and srcName are updated with input values. After calling the export

method, you can call the clearLocal( ) method to indicate the data is stored outside the database and call the deleteContent method if you want to delete the content of the local data.

This method is available for user-defined sources that can support the export method.

The only server-side native support for the export method is for the srcType 'FILE'.

The export( ) method for a source type of 'FILE' is similar to a file copy operation in that the original data stored in the BLOB is not touched other than for reading purposes.

The export( ) method is not an exact mirror operation to the import( ) method in that the clearLocal( ) method is not automatically called to indicate the data is stored oudside the database, whereas the import( ) method automatically calls the setLocal( ) method.

Call the deleteContent method after calling the export( ) method to delete the content from the database if you no longer intend to manage the multimedia data within the database.

The export( ) method only writes to a directory object that the user has privileges to access. That is, you can access a directory that you created using the SQL CREATE DIRECTORY statement or one to which you were granted READ access. To execute the CREATE DIRECTORY statement, you must have the CREATE ANY DIRECTORY privilege. In addition, you must use the DBMS_JAVA.GRANT_ PERMISSION call to specify which files can be written.

For example, the following grants the user, MEDIAUSER, the right to write to the file named filename.dat.

```
CALL DBMS_JAVA.GRANT_PERMISSION(
        'MEDIAUSER',
        'java.io.FilePermission',
        '/actual/server/directory/path/filename.dat',
        'write');
```

See the security and performance section in the *Oracle8i Java Developer's Guide* for more information.

Whenever you import or export images with signature data between platforms, you should use the convert( ) method to ensure that the signature is in the correct format.

Invoking this method implicitly calls the setUpdateTime( ) method.

Examples

Export image data to an external data source, reset the local attribute, and delete the local content.

```
DECLARE
  myImage ORDSYS.ORDVir;
  ctx RAW(4000) := NULL;
BEGIN
  SELECT photo INTO myImage FROM stockphotos WHERE photo_id=1;
  myImage.export(ctx,'FILE','IMAGEDIR','image1.gif';
  myImage.clearLocal;
  myImage.deleteContent;
END;
/
```

getBFILE Method

Format

getBFILE RETURN BFILE;

Description

Returns the LOB locator of the BFILE containing the image.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getBFILE, WNDS, WNPS, RNDS, RNPS)

Exceptions

INCOMPLETE_SOURCE_INFORMATION -- Raised if the srcType is NULL.

INVALID_SOURCE_TYPE -- Raised if srcType is not FILE.

Usage Notes

This method constructs a BFILE using the stored srcLocation and srcName. srcLocation must contain a defined directory object, and srcName must contain a valid file name.

Examples

Get the BFILE for the image.

```
DECLARE
    Image ORDSYS.ORDVir;
    imagebfile BFILE;
BEGIN
    SELECT photo INTO Image FROM stockphotos
      WHERE photo_id = 1;

    -- get the image BFILE
    imagebfile := Image.getBFILE;
END;
```

getCompressionFormat Method

Format

getCompressionFormat RETURN VARCHAR2;

Description

Returns the compression type of an image. This method does not actually read the image, it is a simple accessor method that returns the value of the compressionFormat attribute.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getCompressionFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

Use this method rather than accessing the compressionFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDVir object.

Examples

Get the compression type of an image.

```
imgb1             ORDSYS.ORDVir;
compressionFormat VARCHAR2(4000);

...
compressionFormat := imgb1.getCompressionFormat;
```

getContent Method

Format

getContent RETURN BLOB;

Description

Returns the LOB locator of the BLOB containing the image. This is a simple access method that returns the value of the localData attribute.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContent, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

This method can be used in conjunction with the getMimeType method. For example, a client trying to access image data to put it on a Web-based viewer can call getMimeType and publish the header and then call getContent to access the image data and send it over the wire.

Use this method rather than accessing the content attribute directly, to protect yourself from potential changes to the internal representation of the ORDVir object.

Examples

A client accesses image data.

```
DECLARE
    Image ORDSYS.ORDVir;
    localData BLOB;
BEGIN
    SELECT photo INTO Image FROM stockphotos
      WHERE photo_id = 1;
    -- get the image BLOB
```

```
    localData := Image.getContent;
END;
```

getContentFormat Method

Format

getContentFormat RETURN VARCHAR2;

Description

Returns the format of an image (such as monochrome or 8-bit grayscale). This method does not actually read the image; it is a simple access method that returns the value of the contentFormat attribute.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContentFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

Use this method rather than accessing the contentFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDVir object.

Examples

Get the type of an image.

```
imgb1         ORDSYS.ORDVir;
contentFormat VARCHAR2(4000);

...
contentFormat := imgb1.getContentFormat;
```

getContentLength( ) Method

Format

getContentLength RETURN INTEGER;

Description

Returns the length of the image data content stored in the source. This method does not actually read the image; it is a simple access method that returns the value of the contentLength attribute.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getContentLength, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

Use this method rather than accessing the contentLength attribute directly to protect yourself from potential changes to the internal representation of the ORDVir object.

Examples

Get the length of the image data content stored in the source.

```
imgb1         ORDSYS.ORDVir;
contentLength INTEGER;

...
contentLength := imgb1.getContentLength;
```

getFileFormat( ) Method

Format

getFileFormat RETURN VARCHAR2;

Description

Returns the file type of an image (such as TIFF or JFIF). This method does not actually read the image; it is a simple access method that returns the value of the fileFormat attribute.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getFileFormat, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

Use this method rather than accessing the fileFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDVir object.

Examples

Get the file type of an image.

```
imgb1      ORDSYS.ORDVir;
fileFormat VARCHAR2(4000);

...
fileFormat := imgb1.getFileFormat;
```

getHeight( ) Method

Format

getHeight RETURN INTEGER;

Description

Returns the height of an image in pixels. This method does not actually read the image; it is a simple accessor method that returns the value of the height attribute.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getHeight, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

Use this method rather than accessing the height attribute directly to protect yourself from potential changes to the internal representation of the ORDVir object.

Examples

Get the height of an image.

```
imgb1  ORDSYS.ORDVir;
height INTEGER;

...
height := imgb1.getHeight;
```

getMimeType( ) Method

Format

getMimeType RETURN VARCHAR2;

Description

Returns the MIME (Multipurpose Internet Mail Exchange) type of an image (such as image/jpeg or image/gif). This method does not actually read the image; it is a simple access method that returns the value of the mimeType attribute.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getMimeType, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

Use this method to obtain the MIME type of the image. The MIME type is required by Web browsers along with image content. It tells the Web browser how to interpret the image.

For unrecognizable formats, you must call the setMimeType( ) method and specify the appropriate MIME type.

See Appendix A for the MIME type associated with each supported file type.

Examples

Return the MIME media type.

```
imgb1      ORDSYS.ORDVir;
mimeType   VARCHAR2(4000);
...
mimeType := imgb1.getMimeType;
```

getSignature( ) Method

Format

getSignature RETURN RAW;

Description

Returns the signature of an image. This method does not actually create the image signature; it is a simple access method that returns the contents of the signature attribute.

Parameters

None.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method rather than accessing the signature attribute directly to protect yourself from potential changes to the internal representation of the ORDVir object.

Examples

Get the signature of an image.

```
virb1     ORDSYS.ORDVir;
signature RAW(2000);

...
signature := virb1.getSignature;
```

getSource( ) Method

Format

getSource RETURN VARCHAR2;

Description

Returns information about the external location of the image data in URL format.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSource, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

The possible return values for this method are:

- file:/ / <DIR OBJECT NAME> / <FILE NAME>
- HTTP:/ / <URL>
- <user-defined_type>:/ / <user_defined_location> / <user_defined_name>

Examples

Get the source of the image data.

```
DECLARE
    Image ORDSYS.ORDVir;
    SourceInfo VARCHAR2(4000);
BEGIN
    SELECT photo INTO Image FROM stockphotos
      WHERE photo_id = 1;
    -- get the image source information
    SourceInfo := Image.getSource;
END;
```

getSourceLocation( ) Method

Format

getSourceLocation RETURN VARCHAR2;

Description

Returns a formatted string containing the external image data source location.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceLocation, WNDS, WNPS, RNDS, RNPS)

Exceptions

INCOMPLETE_SOURCE_LOCATION -- Raised if the location is not set in the image object.

Usage Notes

This method returns a VARCHAR2 string formatted as <location> for the current value of the srcLocation attribute.

Examples

Get the source information about an image data source location.

```
DECLARE
    Image ORDSYS.ORDVir;
    SourceLocation VARCHAR2(4000);
BEGIN
    SELECT photo INTO Image FROM stockphotos
      WHERE photo_id = 1;

    -- get the image source location
    SourceLocation := Image.getSourceLocation;
END;
```

getSourceName( ) Method

Format

getSourceName RETURN VARCHAR2;

Description

Returns a string containing the name of the external image data source.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceName, WNDS, WNPS, RNDS, RNPS)

Exceptions

INCOMPLETE_SOURCE_NAME -- Raised if the srcName is not set in the image object.

Usage Notes

This method returns a VARCHAR2 string containing the name of the external data source, for example, 'testimg.dat'.

Examples

Get the source name information about an image data source.

```
DECLARE
    Image ORDSYS.ORDVir;
    SourceName VARCHAR2(4000);
BEGIN
    SELECT photo INTO Image FROM stockphotos
      WHERE photo_id = 1;

    -- get the image source name
    SourceName := Image.getSourceName;
END;
/
```

getSourceType( ) Method

Format

getSourceType RETURN VARCHAR2;

Description

Returns a string containing the type of the external data source.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getSourceType, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

This method returns a VARCHAR2 string containing the type of the external image data source, for example, 'FILE'.

Examples

Get the source type information about an image data source.

```
DECLARE
    Image ORDSYS.ORDVir;
    SourceType VARCHAR2(4000);
BEGIN
    SELECT photo INTO Image FROM stockphotos
      WHERE photo_id = 1;

    -- get the image source type
    SourceType := Image.getSourceType;
END;
/
```

getUpdateTime( ) Method

Format

getUpdateTime RETURN DATE;

Description

Returns the time when the image was last updated.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getUpdateTime, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

None.

Examples

Get the updated time of some image data.

```
DECLARE
   Image       ORDSYS.ORDVir;
   UpdateTime  DATE;
BEGIN
   SELECT photo INTO Image FROM stockphotos
     WHERE photo_id = 1;
 -- get the image update time
   UpdateTime := Image.getUpdateTime;
END;
/
```

getWidth( ) Method

Format

getWidth RETURN INTEGER;

Description

Returns the width of an image in pixels. This method does not actually read the image; it is a simple access method that returns the value of the width attribute.

Parameters

None.

Pragmas

PRAGMA RESTRICT_REFERENCES(getWidth, WNDS, WNPS, RNDS, RNPS)

Exceptions

None.

Usage Notes

Use this method rather than accessing the width attribute directly to protect yourself from potential changes to the internal representation of the ORDVir object.

Examples

Get the width of an image.

```
imgb1  ORDSYS.ORDVir;
width  INTEGER;

...
width := imgb1.getWidth;
```

import( ) Method

Format

```
import(ctx IN OUT RAW);
```

Description

Transfers image data from an external image data source to a local source (localData) within an Oracle database.

Parameters

ctx
Specifies the source plug-in context information. The ctx structure must be allocated, and you must call the source.open( ) method. See the beginning of this chapter for more information.

Pragmas

None.

Exceptions

- INCOMPLETE_SOURCE_INFORMATION --Raised if the value of srcType is 'NULL'.
- NULL_SOURCE -- Raised if the value of BLOB is 'NULL'.
- METHOD_NOT_SUPPORTED -- Raised if the import( ) method is not supported by the source plug-in being used.

Usage Notes

Use the setSource( ) method to set the external source type, location, and name, before calling the import( ) method.

After importing data from an external image data source to a local source (within an Oracle database), the source information remains unchanged (that is, it points to the source from where the data was imported).

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

If the file format of the imported image is not previously set to a string beginning with "OTHER", the setProperties( ) method is also called. Set the file format to a string preceded by "OTHER" for foreign image formats; calling the setProperties( ) method for Foreign Images does this for you.

Whenever you import or export images with signature data between platforms, you should use the convert( ) method to ensure that the signature is in the correct format.

Examples

Import image data from an external image data source into the local source.

```
DECLARE
    Image ORDSYS.ORDVir;
    ctx  RAW(4000) := NULL;
BEGIN
    -- select the image to be imported
    SELECT photo INTO Image FROM stockphotos
           WHERE photo_id = 1 FOR UPDATE;

    -- import the image into the database
    Image.import(ctx);
END;
/
```

importFrom( ) Method

Format

```
importFrom(ctx              IN OUT RAW,
           source_type      IN VARCHAR2,
           source_location  IN VARCHAR2,
           source_name      IN VARCHAR2);
```

Description

Transfers image data from the specified external image data source to a local source (localData) within an Oracle database.

Parameters

ctx
Specifies the source plug-in context information. The ctx structure must be allocated, and you must call the source.open( ) method. See the beginning of this chapter for more information.

source_type
Specifies the source type of the image data.

source_location
Specifies the location from which the image data is to be imported.

source_name
Specifies the name of the image data.

Pragmas

None.

Exceptions

None.

Usage Notes

This method is similar to the import( ) method, except the source information is specified as parameters to the method instead of separately.

After importing data from an external image data source to a local source (within an Oracle database), the source information remains (set to the input parameters) and points to the source from where the data was imported.

Invoking this method implicitly calls the setUpdateTime( ) and setLocal methods.

If the file format of the imported image is not previously set to a string beginning with OTHER, the setProperties( ) method is also called. Set the file format to a string preceded by OTHER for foreign image formats; calling the setProperties( ) method for Foreign Images does this for you.

Whenever you import or export images with signature data between platforms, you should use the convert( ) method to ensure that the signature is in the correct format.

Examples

Import image data from the specified external data source into the local source.

```
DECLARE
    Image ORDSYS.ORDVir;
    ctx   RAW(4000):=NULL;
BEGIN
    -- select the image to be imported
    SELECT photo INTO Image FROM stockphotos
           WHERE photo_id = 1 FOR UPDATE;

    -- import the image into the database
    Image.importFrom(ctx,
                     'FILE',
                     'ORDVIRDIR',
                     'virdemo1.dat');

    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
END;
/
```

isLocal( ) Method

Format

isLocal RETURN BOOLEAN;

Description

Returns the current value of the local flag, indicating whether or not the data is stored locally.

Parameters

None.

Pragmas

Pragma RESTRICT_REFERENCES(isLocal, WNDS, WNPS,RNDS,RNPS)

Exceptions

None.

Usage Notes

A value of TRUE means the data is local to the database. A value of FALSE means the data is external to the database.

Examples

Determine whether or not data is local.

```
DECLARE
    Image ORDSYS.ORDVir;
BEGIN
    SELECT photo INTO Image FROM stockphotos WHERE photo_id = 1;
    -- check to see if image is stored locally
    IF Image.isLocal THEN
        DBMS_OUTPUT.PUT_LINE('Image is stored locally');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Image is stored externally');
    END IF;
END;
```

migrateFromORDVirB( ) Method

Format

migrateFromORDVirB(old_object ORDVirB);

Description

Allows you to migrate old ORDVirB images to the new ORDVir object type.

Parameters

old_object
The old ORDVirB image.

Pragmas

None.

Exceptions

- NULL_SOURCE -- Raised if src(old_object) is NULL.
- NULL_DESTINATION -- Raised if destination ORDVir is NULL.
- NULL_CONTENT -- Raised if old object content attribute is NULL.
- NULL_LOCAL_DATA -- Raised if destination ORDVir source.localData is NULL.

Usage Notes

This method copies images from the source BLOB to the destination BLOB, copies all image attributes from the old object to the new object, and sets the update time and local attribute.

Examples

Migrate an old ORDVirB image to a new ORDVir image.

```
DECLARE
    Image ORDSYS.ORDVir;
    old_image ORDSYS.ORDVirB;
BEGIN
```

```
    -- Select the old image
    SELECT imageb INTO old_image FROM old_images WHERE id = 1;

    -- Select the new image
    SELECT photo INTO Image FROM stockphotos WHERE photo_id = 1 FOR UPDATE;

    -- Migrate from the old to the new
    Image.migrateFromORDVirB(old_image);

    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
END;
/
```

migrateFromORDVirF( ) Method

Format

migrateFromORDVirF(old_object ORDVirF);

Description

Allows you to migrate old ORDVirF images to the new ORDVir object type.

Parameters

old_object
The old ORDVirF image.

Pragmas

None.

Exceptions

None.

Usage Notes

Extracts the directory name and file name from the source image and copies them to the srcLocation and SrcName attributes of the destination image. This method copies all image attributes from the old object to the new object, sets the update time, and clears the local attribute.

Examples

Migrate an old ORDVirF image to a new ORDVir image.

```
DECLARE
    Image ORDSYS.ORDVir;
    old_image ORDSYS.ORDVirF;
BEGIN
    -- Select the old image
    SELECT imagef INTO old_image FROM old_images WHERE id = 1;

    -- Select the new image
```

```
    SELECT photo INTO Image FROM stockphotos WHERE photo_id = 1 FOR UPDATE;

    -- Migrate from the old to the new
    Image.migrateFromORDVirF(old_image);

    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
END;
/
```

process( ) Method

Format

process(command IN VARCHAR2);

Description

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

Parameters

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

Pragmas

None.

Exceptions

DATA_NOT_LOCAL -- Raised if data is not local or source.localData is NULL.

Usage Notes

You can change one or more of the image attributes shown in Table 4–5. Table 4–6 shows additional changes that can be made only to raw pixel and foreign images.

Table 4–5 Image Processing Operators

Operator Name	Usage	Values
compressionFormat	Compression type / format	JPEG, SUNRLE, BMPRLE, TARGARLE, LZW, LZWHDIFF, FAX3, FAX4, HUFFMAN3, Packbits, GIFLZW
compressionQuality	Compression quality	MAXCOMPRATIO, MAXINTEGRITY, LOWCOMP, MEDCOMP, HIGHCOMP
contentFormat	Imagetype / pixel / data format	MONOCHROME, 8BITGRAYSCALE, 8BITGREYSCALE, 8BITLUT, 24BITRGB

Table 4–5 Image Processing Operators (Cont.)

Operator Name	Usage	Values
cut	Window to cut or crop (origin.x origin.y width height)	(INTEGER INTEGER INTEGER INTEGER) Maximum value is 65535
fileFormat	File format of the image	BMPF, CALS, GIFF, JFIF, PICT, RASF, RPIX, TGAF, TIFF
fixedScale	Scale to a specific size in pixels (Width, Height)	(INTEGER INTEGER)
maxScale	Scale to a specific size in pixels, while maintaining the aspect ratio (maxWidth, maxHeight)	(INTEGER INTEGER)
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 4–6 Additional Image Processing Operators for Raw Pixel and Foreign Images

Operator Name	Usage	Values
ChannelOrder	Indicates the relative position of the red, green, and blue channels (bands) within the image.	RGB (default), RBG, GRB, GBR, BRG, BGR
InputChannels	For multiband images, specify either one (grayscale) or three integers indicating which channels to assign to red (first), green (second), and blue (third). Note that this parameter affects the source image, not the destination.	INTEGER or INTEGER INTEGER INTEGER
Interleave	Controls band layout within the image: Band Interleaved by Pixel Band Interleaved by Line Band Sequential	BIP (default), BIL, BSQ
PixelOrder	If NORMAL, then the leftmost pixel appears first in the image.	NORMAL (default), REVERSE
ScanlineOrder	If NORMAL, then the top scanline appears first in the image.	NORMAL (default), INVERSE

Note: When specifying 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.

There is no implicit import( ) or importFrom( ) call performed when you call this method; if data is external, you must first call import( ) or importFrom( ) to make the data local before you can process it.

Implicit setProperties( ), setUpdateTime( ), and setMimeType( ) methods are performed after the process( ) method is called.

Note that because you are changing the object, you must have the appropriate write access to the table. Also, you must select the object for update.

See Appendix C for complete descriptions of the process( ) operators.

Examples

Example 1: Change the file format of image1 to GIF.

```
SELECT photo INTO image1 FROM stockphotos where photo_id = 1 FOR UPDATE
image1.process('fileFormat=GIFF');
```

Example 2: 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"');
```

Note that changing the length on only one axis (for example, xScale=2.0) does not affect the length on the other axis, and would result in image distortion. Also, only the xScale and yScale parameters can be combined in a single operation. Any other combinations of scaling operators result in an error.

Example 3: The maxScale and fixedScale operators are especially useful for creating thumbnail images for use on a Web site from various-sized originals. The following line creates a 32-by-32 pixel thumbnail image, preserving the original aspect ratio.

```
image1.process('maxScale= 32 32');
```

processCopy( ) Method

Format

```
processCopy(command IN VARCHAR2,
            dest    IN OUT NOCOPY  ORDVir);
```

Description

Copies an image stored internally or externally to another image stored internally as a BLOB.

Parameters

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

dest
The destination of the new image.

Pragma

None.

Exceptions

NULL_DESTINATION -- Raised if the value of dest is NULL.

NULL_LOCAL_DATA -- Raised if source.localData is NULL and source.isLocal is 1. Also raised if dest.source.localData is NULL.

DATA_NOT_LOCAL -- Raised if the destination image is not local.

Usage Notes

- See Table 4–5, "Image Processing Operators" and Table 4–6, "Additional Image Processing Operators for Raw Pixel and Foreign Images". See Appendix C for complete descriptions of the processCopy( ) operators.
- You cannot specify the same BLOB as both the source and the destination.
- Calling this method processes the image into the destination BLOB from any source (local or external).

- The destination source must be initialized before calling this method.
- The destination image must be local.
- Implicit setProperties( ), setUpdateTime( ), and setMimeType( ) methods are done after the processCopy( ) method is called on the destination image.

Examples

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

```
create or replace procedure copyit is
  imgB1     ORDSYS.ORDVir;
  imgB4     ORDSYS.ORDVir;
  mycommand   VARCHAR2(400);
begin
  select photo into imgB1 from stockphotos  where photo_id = 1;
  select photo into imgB4 from stockphotos where photo_id = 4 for update;
  mycommand:= 'fileFormat=tiff compressionFormat = packbits
  contentFormat = 8bitlut';
  imgB1.processcopy(mycommand,imgB4);
  update stockphotos set photo = imgB4 where photo_id = 4;
end;
```

setLocal Method

Format

setLocal;

Description

Sets the local attribute to indicate that the data is stored internally in a BLOB. When local is set, image methods look for image data in the source.localData attribute.

Parameters

None.

Pragmas

None.

Exceptions

NULL_LOCAL_DATA -- Raised if source.localData is NULL.

Usage Notes

This method sets the local attribute to 1, meaning the data is stored locally in localData.

Examples

Indicate that the image is stored locally.

```
DECLARE
    Image ORDSYS.ORDVir;
BEGIN
    SELECT photo INTO Image FROM stockphotos WHERE photo_id = 1 FOR UPDATE;

    -- set local so we look for the image in the database
    Image.setLocal;

    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
END;
/
```

setMimeType( ) Method

Format

setMimeType(mime IN VARCHAR2);

Description

Allows you to set the MIME media type of the stored image data.

Parameters

mime
The MIME media type.

Pragmas

None.

Exceptions

None.

Usage Notes

You can override the automatic setting of MIME information by calling this method with a specified MIME value.

Visual Information Retrieval cannot automatically set the MIME type of foreign images. You must use this method to set the MIME type for foreign images.

Examples

Set the MIME media type for some stored image data.

```
DECLARE
    Image ORDSYS.ORDVir;
BEGIN
    SELECT photo INTO Image FROM stockphotos FOR UPDATE
      WHERE photo_id = 1;
     -- set the image mime type
    Image.setMimeType('image/bmp');
END;
```

setProperties( ) Method

Format

setProperties;

Description

Reads the image data to get the values of the object attributes, then stores them in the appropriate attribute fields. The image data can be stored in the database in a BLOB, or externally in a BFILE or URL. If the data is stored externally in anything other than a BFILE, the data is read into a temporary BLOB to determine image characteristics.

This method should not be called for foreign images. Use the setProperties(description) method for foreign images.

Parameters

None.

Pragmas

None.

Exceptions

NULL_LOCAL_DATA -- Raised if the source.localData attribute is NULL and the source.local attribute indicates the data is local.

Usage Notes

After you have copied, stored, or processed a native format image, call this method to set the current characteristics of the new content. Note that some methods already call setProperties( ) implicitly.

This procedure sets the following information about an image:

- Height in pixels
- Width in pixels
- Data size of the image in bytes
- File type (TIFF, JFIF, and so forth)

- Image type (monochrome, 8-bit grayscale, and so forth)
- Compression type (JPEG, LZW, and so forth)
- MIME type (generated based on the file format)

This method implicitly sets the update time and MIME type.

Examples

Select the image, and then set the attributes using the setProperties method.

```
DECLARE
    imgB1 ORDSYS.ORDVir;
BEGIN
    select photo into imgB1 from stockphotos where photo_id = 1 for update;
    imgB1.setProperties;
    dbms_output.put_line('image width = '|| imgB1.getWidth );
    dbms_output.put_line('image height = '|| imgB1.getHeight );
    dbms_output.put_line('image size = '|| imgB1.getContentLength );
    dbms_output.put_line('image file type = '|| imgB1.getFileFormat );
    dbms_output.put_line('image type = '|| imgB1.getContentFormat );
    dbms_output.put_line('image compression = '|| imgB1.getCompressionFormat )
    dbms_output.put_line('image MIME type = '|| imgB1.getMimeType );
END;
/
```

Example output:

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

setProperties( ) Method for Foreign Images

Format

```
setProperties(description IN VARCHAR2);
```

Description

Allows you to write the characteristics of a foreign image (one not natively supported) into the appropriate attribute fields.

Parameters

description
The image characteristics to set for the foreign image.

Pragmas

None.

Exceptions

NULL_PROPERTIES_DESCRIPTION -- Raised if the description is NULL.

Usage Notes

Note: Once you have set the properties for a foreign image, it is up to you to keep the properties consistent. If Visual Information Retrieval detects an unknown file format, it will not implicitly set the properties.

After you copied, stored, or processed a foreign image, call this method to set the characteristics of the new image content. Unlike the native image types described in Appendix A, foreign images either do not contain information on how to interpret the bits in the file or Visual Information Retrieval does not understand the information. In this case, you must set the information explicitly.

You can set the following image characteristics for foreign images, as shown in Table 4–7.

Table 4–7 **Image Characteristics for Foreign Images**

Field	Data Type	Description
CompressionFormat	STRING	Value must be CCITG3, CCITG4, or NONE (default).
DataOffset	INTEGER	The offset allows the image to have a header that Visual Information Retrieval does not try to interpret. Set the offset to avoid any potential header. The value must be a positive integer less than the LOB length. Default is zero.
DefaultChannelSelection	INTEGER	For multiband images, specify either one (grayscale) or three integers indicating which channels to assign to red (first), green (second), and blue (third). For example, DefaultChannelSelection = 1 for single band images and DefaultChannelSelection = 1, 2, 3 for triple-band images.
Height	INTEGER	Height of the image in pixels. Value must be a positive integer. There is no default, and a value must be specified.
Interleaving	STRING	Band layout within the image. Valid styles are: ■ BIP (default): Band Interleaved by Pixel ■ BIL: Band Interleaved by Line ■ BSQ: Band Sequential
NumberOfBands	INTEGER	Value must be a positive integer less than 255 describing the number of color bands in the image. Default is 3.
PixelOrder	STRING	If NORMAL (default), the leftmost pixel appears first in the file. If REVERSE, the rightmost pixel appears first.
ScanlineOrder	STRING	If NORMAL (default), the top scanline appears first in the file. If INVERSE, then the bottom scanline appears first.
UserString	STRING	A 4-character descriptive string. If used, the string is stored in the fileFormat field, appended to the file format (OTHER:). Default is blank and fileFormat is set to OTHER.
Width	INTEGER	Width of the image in pixels. Value must be a positive integer. There is no default, and a value must be specified.
MimeType	STRING	Value must be a MIME type, such as img/gif.

The values supplied to setProperties( ) are written to the existing ORDVir data attributes. The fileFormat is set to OTHER and includes the user string, if supplied, for example, OTHER: DICOM.

Examples

Select the foreign image and then set the properties for the image.

```
DECLARE
    Image ORDSYS.ORDVIR;
BEGIN
    SELECT photo INTO Image FROM stockphotos
           WHERE photo_id = 1 FOR UPDATE;
    -- set property attributes for the image data
    Image.setProperties('width=123 height=321 compressionFormat=NONE' ||
                        ' userString=DJM dataOffset=128' ||
                        ' scanlineOrder=INVERSE pixelOrder=REVERSE' ||
                        ' interleaving=BIL numberOfBands=1' ||
                        ' defaultChannelSelection=1');
    UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
END;
/
```

setSource( ) Method

Format

```
setSource(source_type       IN VARCHAR2,
          source_location   IN VARCHAR2,
          source_name       IN VARCHAR2);
```

Description

Sets or alters information about the external source of the image data.

Parameters

source_type
Sets or alters the srcType attribute of the image data source.

source_location
Sets or alters the srcLocation attribute of the image data source.

source_name
Sets or alters the srcName attribute of the image data source.

Pragmas

None.

Exceptions

None.

Usage Notes

Users can use this method to set the image data source to a new external file or URL.

Calling this method implicitly calls the setUpdateTime( ) method.

Examples

Set the source of the image data.

```
DECLARE
   Image ORDSYS.ORDVir;
   BEGIN
      SELECT photo INTO Image FROM stockphotos FOR UPDATE
         WHERE photo_id = 1;
   -- set source information for the image
      Image.setSource('file', 'ORDVIRDIR', 'jdoe.gif');
      UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
   END;
/
```

setUpdateTime( ) Method

Format

setUpdateTime(current_time DATE);

Description

Sets the time when the image data was last updated.

Parameters

current_time
The updated time. Default is SYSDATE.

Pragmas

None.

Exceptions

None.

Usage Notes

Use this method whenever you modify the image data. The process( ), processCopy( ), and setProperties( ) methods all call this method and setMimeType( ) implicitly.

Examples

Set the updated time of some image data.

```
DECLARE
   Image ORDSYS.ORDVir;
   BEGIN
      SELECT photo INTO Image FROM stockphotos FOR UPDATE
         WHERE photo_id = 1;
   -- set image update time
      Image.setUpdateTime(SYSDATE);
      UPDATE stockphotos SET photo = Image WHERE photo_id = 1;
   END;
/
```

4.5 Operators for Visual Information Retrieval

The Visual Information Retrieval operators are located in the ORDSYS schema.

Two operators are located in the ORDSYS.VIR package:

- Analyze( ): Produces the signature of an image.
- Convert( ): Converts the signature to either big or little endian byte order based on the architecture of the host machine.

For ease of use, you can create a local synonym for the ORDSYS.ORDVIR_PKG package. Connect to your schema and enter the following command:

```
SQL> CREATE SYNONYM VIR FOR ORDSYS.ORDVIR_PKG;
```

After creating the synonym, you can use it in calls to the operators: VIR.Analyze( ), VIR.Convert( ), and so forth. Note that you must have the default CREATE SYNONYM privilege.

Two additional operators are schema level operators and do not reside within the package. These operators use the domain index, if it exists.

- ORDSYS.VIRSimilar( ): Compares two signatures and determines whether or not the images match, based on the weights and threshold. Returns 1 if the computed distance measure (weighted average) is less than or equal to the threshold value, and otherwise returns 0.
- ORDSYS.VIRScore( ): Returns the score value computed by the primary operator, VIRSimilar( ). VIRScore( ) is an ancillary operator to VIRSimilar( ).

Analyze( ) Operator

Format

Analyze(image IN OUT ORDVir);

Description

Analyzes an image, derives information relating to the visual attributes, creates the image signature, and stores it in the ORDVir object.

Parameters

image
The ORDVir object to be analyzed.

Pragmas

None.

Exceptions

None.

Usage Notes

The Analyze( ) operator creates the image signature (or feature vector), which is necessary for any content-based retrieval. Whenever you are working with a new or changed image, you should call Analyze( ) to generate a signature and then use the setProperties( ) method to set the other image characteristics.

During analysis, images are temporarily scaled to a common size such that the signatures are based on a common reference.

Example

Create the signatures for all images in the stockphotos table.

```
DECLARE
   temp_image   ORDSYS.ORDVir;
   temp_id      INTEGER;
   cursor c1 is select photo_id, photo from stockphotos for update;
BEGIN
```

```
   OPEN c1;
   LOOP
      fetch c1 into temp_id, temp_image;
      EXIT WHEN c1%NOTFOUND;

 -- Generate signature and set the properties for the image.
      ORDSYS.VIR.Analyze(temp_image);
      temp_image.setProperties;
      UPDATE stockphotos SET photo = temp_image WHERE photo_id = temp_id;
   END LOOP;
   CLOSE c1;
END;
```

Convert( ) Operator

Format

```
Convert(signature IN OUT RAW, operations IN VARCHAR2)
    RETURN  BINARY_INTEGER;
```

Description

Converts the image signature to a format usable by the host machine.

Parameters

signature
The signature of the image, as created by the Analyze( ) operator. Data type is RAW(2000).

operations
The kind of processing done to the image signature. The following operation is available:

Operation Keyword	Description
BYTEORDER	Converts the signature to the natural byte order of the host machine.

Pragmas

None.

Exceptions

None.

Usage Notes

When the operation is BYTEORDER, the signature is converted to the format of the host machine regardless of its initial state.

This procedure is useful if the database is stored on a remote system, but you want to do your processing locally. If your host machine is from Sun Microsystems, Inc., the Convert( ) operator sets the signature to the big endian byte order. On an Intel

Corporation machine, the operator converts the signature to the little endian byte order. Note that the images themselves are platform-independent; only the signatures need to be converted.

Examples

Convert the signature of the image with `photo_id=1` to the platform of the host system.

```
DECLARE
  myimage ORDSYS.ORDVir;
  convert_result BINARY_INTEGER;
BEGIN
  SELECT photo INTO myimage FROM stockphotos WHERE photo_id=1 FOR UPDATE;
  convert_result := ORDSYS.VIR.Convert(myimage.signature,'BYTEORDER');
  UPDATE stockphotos SET photo=myimage WHERE photo_id =1;
END;
```

VIRScore( ) Operator

Format

VIRScore(referencetoSimilar IN NUMBER);

Description

Compares the signatures of two images and returns a number representing the weighted sum of the distances for the visual attributes. VIRScore( ) is an ancillary operator, used only in conjunction with the primary operator, VIRSimilar( ).

Parameters

referencetoSimilar
The corresponding invocation of VIRSimilar. If there are multiple invocations of VIRSimilar in the same query, this parameter is used to maintain reference. Data type is NUMBER.

Return Value

This function returns a FLOAT value between 0.0 and 100.0, where 0.0 is identical and 100.0 is completely different.

Pragmas

None.

Exceptions

None.

Usage Notes

Before the VIRScore( ) operator can be used, the image signatures must be created with the VIR.Analyze( ) operator. Also, if you want the comparison to use the domain index, the index of type ORDVIRIDX must have already been created. See Section 2.4 for information on creating and using the index, and see Section 3.8 for additional performance tips.

The VIRScore( ) operator can be useful when an application wants to make finer distinctions about matching than the simple Yes or No returned by

VIRSimilar( ). For example, using the number for weighted sum returned by VIRScore( ), the application might assign each image being compared to one of several categories, such as Definite Matches, Probable Matches, Possible Matches, and Nonmatches. The VIRScore( ) operator can also be useful if the application needs to perform special processing based on the degree of similarity between images.

The VIRScore( ) operator also works with old signatures stored using the deprecated ORDVirB and ORDVirF object types. However, Oracle Corporation recommends that you migrate to the new object type using the migrateFromORDVirF( ) and migrateFromORDVirB( ) methods.

Examples

Find the weighted sum of the distances between a test image and the other images in the stockphotos table, using a threshold of 50 and the following weights for the visual attributes:

- Global color: 0.2
- Local color: 0.2
- Texture: 0.1
- Structure: 0.5

This example assumes that the signatures were already created using the Analyze( ) operator and they are stored in the database. Notice that both VIRScore( ) and VIRSimilar( ) are using 123 as the reference number in this example.

```
DECLARE
   img_score       NUMBER;
   i               INTEGER;
   query_signature RAW(2000);
   img             ORDSYS.ORDVir;

BEGIN
SELECT photo_id, ORDSYS.VIRScore(123), photo INTO i, img_score, img FROM stockphotos WHERE
     ORDSYS.VIRSimilar(img.signature, query_signature,
               'globalcolor="0.2" localcolor="0.2"
                 texture="0.1" structure="0.5"', 50, 123) = 1
END;
```

The following shows possible results from this example. The first image has the lowest score, and therefore is the best match of the test image. Changing the weights used in the scoring would lead to different results.

```
PHOTO_ID   SCORE
---------- ------------
         2    3.79988
         4    47.8139
  2 rows selected.
```

VIRSimilar( ) Operator

Format

```
VIRSimilar(signature          IN RAW,
           querysignature     IN RAW,
           weightstring       IN VARCHAR2,
           threshold          IN FLOAT
           [ ,referencetoScore  IN NUMBER]  );
```

where weightstring is: 'globalcolor="val" localcolor="val" texture="val" structure="val"'

Description

Determines whether or not two images match. Specifically, the operator compares the signatures of two images, computes a weighted sum of the distance between the two images using user-supplied weight values for the visual attributes, compares the weighted sum with the threshold value, and returns the integer value 1 if the weighted sum is less than or equal to the threshold value. Otherwise, the operator returns 0.

Parameters

signature
The signature of the image to which you are comparing the query image. Data type is RAW(2000). To use the domain index for the comparison, this first parameter must be the signature column on which the domain index has been created. Otherwise, Oracle8*i* uses the non-indexed implementation of query evaluation.

querysignature
The signature of the query or test image. Data type is RAW(2000).

weightstring
A list of weights to apply to each visual attribute. Data type is VARCHAR2. The following attributes can be specified, with a value of 0.0 specifying no importance

and a value of 1.0 specifying highest importance. You must specify a value greater than zero for at least one of the attributes.

Attribute	Description
globalcolor	The weight value (0.0 to 1.0) assigned to the global color visual attribute. Data type is number. Default is 0.0.
localcolor	The weight value (0.0 to 1.0) assigned to the local color visual attribute. Data type is number. Default is 0.0.
texture	The weight value (0.0 to 1.0) assigned to the texture visual attribute. Data type is number. Default is 0.0.
structure	The weight value (0.0 to 1.0) assigned to the structure visual attribute. Data type is number. Default is 0.0.

Note: When specifying parameter values that include floating-point numbers, you should 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.

threshold
The threshold value with which the weighted sum of the distances is to be compared. If the weighted sum is less than or equal to the threshold value, the images are considered to match. This range of this parameter is from 0.0 to 100.0.

referencetoScore
An optional parameter used when ancillary data (score of similarity) is required elsewhere in the query. Set this parameter to the same value here as used in the VIRScore( ) operator. Data type is NUMBER.

Return Value

Returns an integer value of 0 (not similar) or 1 (match).

Pragmas

None.

Exceptions

None.

Usage Notes

Before the VIRSimilar( ) operator can be used, the image signatures must be created with the Analyze( ) operator. Also, to use the domain index, the index of type ORDVIRIDX must have already been created. See Section 2.4 for information on creating and using the index and see Section 3.8 for additional performance tips.

The VIRSimilar( ) operator is useful when the application needs a simple Yes or No for whether or not two images match. The VIRScore( ) operator is useful when an application wants to make finer distinctions about matching or to perform special processing based on the degree of similarity between images.

The weights supplied for the four visual attributes are normalized prior to processing such that they add up to 100 percent. To avoid confusion and meaningless results, you should develop a habit of always using the same scale, whether 0 to 100 or 0.0 to 1.0.

You must specify at least one of the four image attributes in the weightstring.

The VIRSimilar( ) operator also works with old signatures stored using the deprecated ORDVirB and ORDVirF object types. However, Oracle Corporation recommends that you migrate to the new object type using the migrateFromORDVirF( ) and migrateFromORDVirB( ) methods.

Examples

Using the VIR index, find all images similar to the query image using a threshold value of 25 and the following weights for the visual attributes:

- Global color: 20
- Local color: 20
- Texture: 10
- Structure: 50

This example assumes you already used the Analyze( ) operator to generate a signature for the query image. If an index exists on the signature column, it will be used automatically.

```
DECLARE
   t_img ORDSYS.ORDVIR;
   i     INTEGER;
   query_signature RAW(2000);
BEGIN
   SELECT photo_id, photo INTO i, t_img FROM stockphotos WHERE
        ORDSYS.VIRSimilar(t_img.signature, query_signature,
                 'globalcolor="20"  localcolor="20"
                    texture="10" structure="50"', 25) = 1;
END;
```

A

File and Compression Formats

A.1 Supported File and Compression Formats

The following tables describe the file and compression formats supported by Visual Information Retrieval.

To use these tables, find the data format in which you are interested, and then determine the supported formats. For example, Table A–1 shows that Visual Information Retrieval 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
Mime: image / bmp	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

Table A–2 CALS Raster Data Format

Format	Pixel Format	Support
CALS Raster File Format: 'CALS' File Ext: .cal Mime: image/x-ora-cals	Monochrome	Read/Write
	Compression Format	**Support**
	FAX4 (CCITT G4)	Read/Write
	Data Description	**Support**
	NA	NA

Table A–3 EXIF Data Format

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

Table A–4 GIF Data Format

Format	Pixel Format	Support
GIF File Format: 'GIFF' File Ext: .gif Mime: image/gif NOTE: *inter*Media image has limited support for animated GIF images. There is setProperties( ) support; however, processing using the process( ) and processCopy( ) (or Analyze) methods is not supported.	Monochrome 8-bit LUT	Read Read/Write
	Compression Format	**Support**
	GIFLZW (LZW)	Read/Write
	Data Description	**Support**
	NA	NA

Table A–5 JFIF Data Format

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

Table A–6 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
Mime: image/x-ora-pcxf	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

Table A–7 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
Mime: image/pict	16-bit RGB	Read
	24-bit RGB	Read/Write
	Compression Format	**Support**
Choose one of these compression formats:	Packbits	Read/Write
	JPEG (8-bit grayscale and RGB)	Read/Write
	Data Description	**Support**
Choose one or more of these content formats:	Vector/object graphics	Not supported

Table A–8 Raw Pixel Data Format

Format	Pixel Format	Support
Raw Pixel	Monochrome	Read / Write
File Format: 'RPIX'	8-bit grayscale	Read / Write
File Ext: .rpx	24-bit RGB	Read / Write
Mime: image / x-ora-rpix		
	Compression Format	**Support**
Choose one of these compression formats:	Uncompressed	Read / Write 8-bit grayscale and RGB
	FAX3 (CCITT G3)	Read / Write monochrome only
	FAX4 (CCITT G4)	Read / Write monochrome only
	Data Description	**Support**
	Inverse scanline order	Read / Write
	Reverse pixel order	Read / Write
	BIP, BIL, or BSQ interleave	Read / Write
	Alternative band order	Read / Write
	>3 bands	Read

Table A–9 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
Mime: image / x-ora-rasf		
	Compression Format	**Support**
Choose one of these compression formats:	Uncompressed	Read / Write
	SUNRLE (RLE)	Read / Write

Table A–9 Sun Raster Data Format (Cont.)

	Data Description	**Support**
	NA	NA

Table A–10 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
Mime: image / x-ora-tgaf	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

Table A–11 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
Mime: image / tiff	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 grayscale & RGB)	Read / Write
	Data Description	**Support**
Choose one or more of these content formats:	Planar data	Not supported
	Tiled data	Read
	Photometric interpretation	Read / Write
	MSB / LSB	Read / Write

B

Sample Program

A sample program is included with Visual Information Retrieval to demonstrate how to load two images into the database, generate their signatures, and then compare their signatures using a weighted similarity function.

This program uses two data files, virdemo1.dat and virdemo2.dat, as its input. No other input or parameters are required.

Environment

The following assumptions are made:

- Visual Information Retrieval was installed and PUBLIC has EXECUTE privilege on it.
- VIRDEMODIR directory was created and set to `<ORACLE_HOME>/ord/vir/demo` and granted PUBLIC READ access in order that the image data file can be read into the database.
- virdemo1.dat and virdemo2.dat are valid image files that reside in the VIRDEMODIR directory and the user has read / write access to the directory. These two files are installed in the demo directory.
- User SCOTT has the default TIGER password. You may need to increase the tablespace allocated to Scott to successfully run this sample program.

Running the Sample Program

There are two ways to run the sample program: using the included sample images or using your own images.

Example B–1 runs the sample program using the included image files. The images are compared using equal attribute weights:

- Globalcolor = 1.0

- Localcolor = 1.0
- Texture = 1.0
- Structure = 1.0

Example B–1 Run the Sample Program with Included Images

```
% virdemo
Image 1 and 2 have a similarity score of 0.0
```

Example B–2 shows how to specify your own images on the command line. The images must reside in the directory pointed to by VIRDEMODIR, which is the <ORACLE_HOME> / ord / vir / demo directory.

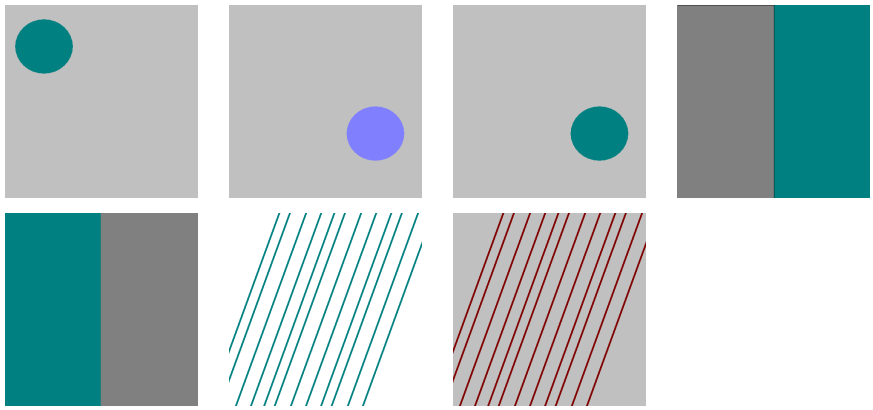
Example B–2 Run the Sample Program with Your Own Images

```
% virdemo <image1> <image2> <global_color> <local_color> <texture> <structure>
```

All six parameters: the 2 file names and 4 attribute weights (ranging from 0.0 to 1.0) must be specified in this sample program. Note that when using the VIRScore( ) operator in your own applications, it is only necessary to provide at least one attribute weight.

Several other sample image files have been provided in the `<ORACLE_HOME>/ord/vir/demo` directory to demonstrate the effects of emphasizing the different visual attributes. You can use an image viewer (such as xv) to display the images, and then compare them using the sample program, experimenting with different weights. Figure B–1 shows the sample images.

Figure B–1 Sample Images in VIRDEMODIR



Understanding the Results

The relative distance between the images being compared is called the score and is output during the execution of the sample program. A lower score indicates a closer match. A score of zero indicates a perfect match.

A score is valid only for a given set of weights for the four visual attributes. If you change the weights, the results will change. For example, consider a comparison of national flags. The flag of Cote d'Ivoire (the Ivory Coast) is composed of three equal vertical color stripes: orange, white, and green. The flag of Ireland is composed of three vertical stripes in the reverse order: green, white, and orange. A comparison of flag images based on global color or structure would consider these two flags identical. However, a comparison emphasizing local color would return a much larger distance between the two images, indicating a poor match.[1]

Sample Program Source Code

The complete C language source code for the sample application is available in the <ORACLE_HOME>/ord/vir/demo directory after installing Visual Information Retrieval. Note that the file specification may differ on your platform.

A small section of the program related to comparing images has been included here.

[1] In the case of using Visual Information Retrieval for flag recognition, you need to know which direction the flag was facing in the test images.

```
.
.
.
 /*
 * ---------------------------------------------------------------------
 * test_1_vir = Analyze the contents of the BLOB and BFILE and return the
 * score based on the weights assigned to the different features
 * ---------------------------------------------------------------------
 */
sb4 test_1_vir(char *g_c, char * l_c, char * txtr, char * stre)
{
  float w_sum = -1.999; /* dummy value */
  text *sqlstmt = (text *)
    "DECLARE \
A            ORDSYS.ORDVir; \
B            ORDSYS.ORDVir; \
a_data       BFILE; \
b_data       BLOB; \
a_sig        RAW(2000);\
b_sig        RAW(2000);\
BEGIN \
SELECT FILEIMAGE INTO A FROM VIRDEMOTAB WHERE C1=1 FOR UPDATE;\
SELECT BLOBIMAGE INTO B FROM VIRDEMOTAB WHERE C1=2 FOR UPDATE;\
a_data := A.getBfile();\
b_data := B.getContent();\
ORDSYS.VIR.Analyze(a_data, a_sig);\
ORDSYS.VIR.Analyze(b_data, b_sig);\
:weighted_sum := ORDSYS.VIR.score(a_sig, b_sig, 'globalcolor='|| :g_color || ',
localcolor= '|| :l_col
or || ', texture=' || :texture || ', structure=' || :strct);\
END;";
```

C

Process and ProcessCopy Operators

This appendix describes the command options, or operators, used in the process and processCopy methods.

The available operators fall into three broad categories, each described in its own section. Those categories are Section C.2, "Image Formatting Operators", Section C.3, "Image Processing Operators", and Section C.4, "Format-Specific Operators". An additional section describes the relative order of these operators.

C.1 Common Concepts

This section describes concepts common to all the image operators and the process( ) and processCopy( ) methods.

C.1.1 Source and Destination Images

The process( ) and processCopy( ) methods operate on one image, called the source image, and produce another image, called the destination image. In the case of the process( ) method, the destination image is written into the same storage space as the source image, replacing it permanently. For the processCopy( ) method, the storage for the destination image is distinct from the storage for the source image.

C.1.2 Process and ProcessCopy

The process( ) and processCopy( ) methods are functionally identical except for the fact that process( ) writes its output into the same BLOB from which it takes its input while processCopy( ) writes its output into a different BLOB. Their command string options are identical, and no distinction is drawn between them.

For the rest of this appendix, the names process( ) and processCopy( ) are used interchangeably, and the use of the name process implies both process( ) and processCopy( ), unless explicitly noted otherwise.

C.1.3 Operator and Value

All the process( ) operators appear in the command string in the form <operator> = <value>. No operator takes effect merely by being present in the command string. The right side of the expression is called the value of the operator, and determines how the operator will be applied.

C.1.4 Combining Operators

In general, any number of operators can be combined in the command string passed into the process( ) method if the combination makes sense. However, certain operators are only supported if other operators are present or if other conditions are met. For example, the compressionQuality operator is only supported if the compression format of the destination image is JPEG. Other operators require that the source or destination image be a raw pixel image or a foreign image.

The flexibility in combining operators allows a single operation to change the format of an image, reduce or increase the number of colors, compress the data, and cut and / or scale the resulting image. This is highly preferable to making multiple calls to do each of these operations sequentially.

C.2 Image Formatting Operators

At the most abstract level, the image formatting operators are used to change the layout of the data within the image storage. They do not change the semantic content of the image, and unless the source image contains more information than the destination image can store, they do not change the visual appearance of the image at all. Examples of a source image with more information than the destination image can store are:

- Converting a 24-bit image to an 8-bit image (too many bits per pixel)
- Converting a color image to a grayscale or monochrome image (too many color planes)
- Converting an uncompressed image, or an image stored in a lossless compression format to a lossy compression format (too much detail)

C.2.1 FileFormat

The **FileFormat** operator determines the image file type, or format, of the output image. The value of this operator is a 4-character code that is a mnemonic for the new file format name. The list of allowable values for the file format operator is shown in Table 4–5. Appendix A contains basic information about each file format, including its mnemonic, typical file extension, allowable compression and content formats, and other notable features.

The value given to the file format operator is the single most important detail when specifying the output for process( ). This value determines the range of allowable content and compression formats, whether or not compression quality will be useful, and whether or not the format specific operators will be useful.

If the file format operator is not used in the process( ) command string, Visual Information Retrieval will determine the file format of the source image and use that as the default file format value. If the file format of the source image does not support output, then an error will occur. If the source image is a foreign image then the output image will be written as raw pixel data.

C.2.2 ContentFormat

The **ContentFormat** operator determines the format of the image content. The content is taken to mean the number of colors supported by the image and the manner in which they are supported. Depending on which file format is used to store the output image, some or most of the content formats may not be supported.

The supported values for the ContentFormat operator fall into three broad classes with three additional special values. The actual mnemonics for these values are listed in Table 4–5.

The content formats whose names include grayscale or greyscale support only shades of gray. The differences between these content formats is how many shades are allowed. The 4bit formats support 16 shades, while the 8bit formats support 256 shades of gray. There is no distinction between grayscale and greyscale.

The content formats whose names include LUT use a color look-up table to support various colors. The 1bitlut format allows two distinct colors; the 2bitlut format supports 4 colors; the 4bitlut format supports 16 unique colors; and the 8bitlut format supports 256 colors.

The content formats whose names include RGB store the color values directly in the pixel data as a red, green, blue triplet. The number of bits of total RGB data is specified in the format name, and individual formats allocate these bits to red, green, and blue in different ways. However, more bits of data allow for finer

distinctions between different shades in any case. Not all bits are used by some image formats.

The monochrome content format allows only black and white to be stored, with no gray shades in between. The raw and 24bitplanar formats are not currently supported.

If the ContentFormat operator is not passed to the process( ) method, then Visual Information Retrieval attempts to duplicate the content format of the source image, if it is supported by the file format of the destination image. Otherwise, a default content format is chosen depending on the destination file format.

C.2.3 CompressionFormat

The **CompressionFormat** operator determines the compression algorithm used to compress the image data. The range of supported compression formats depends heavily upon the file format of the output image. Some file formats support but a single compression format, and some compression formats are only supported by one file format.

The supported values for the CompressionFormat operator are listed in Table 4–5.

All compression formats that include RLE in their mnemonic are run-length encoding compression schemes, and work well only for images that contain large areas of identical color. The PACKBITS compression type is a run-length encoding scheme that originates from the Macintosh system, but it is supported by other systems. It has limitations that are similar to other run-length encoding compression formats. Formats that contain LZW or HUFFMAN are more complex compression schemes that examine the image for redundant information, and they are more useful for a broader class of images. FAX3 and FAX4 are the CCITT Group 3 and Group 4 standards for compressing facsimile data and are useful only for monochrome images. All the compression formats mentioned in this paragraph are lossless compression schemes, which means that compressing the image does not discard data. An image compressed into a lossless format and then decompressed will look the same as the original image.

The JPEG compression format is a special case. Developed to compress photographic images, the JPEG format is a lossy format, which means that it compresses the image by discarding unimportant details. Because this format is optimized for compressing photographic and similarly complex images, it often produces poor results for other image types, such as line art images and images with large areas of similar color. JPEG is the only lossy compression scheme currently supported by Visual Information Retrieval.

If the CompressionFormat operator is not used, then Visual Information Retrieval will attempt to use the compression format of the source image if it is supported by the destination image file and content format. Otherwise, a default compression scheme is used depending on the destination file format. This default scheme is often none or no compression.

C.2.4 CompressionQuality

The **CompressionQuality** operator determines the relative quality of an image compressed with a lossy compression format. This operator has no meaning for lossless compression formats, and hence is not currently supported for any compression format except JPEG.

The CompressionQuality operator accepts five values, ranging from most compressed image (lowest visual quality) to least compressed image (highest visual quality): MAXCOMPRATIO, HIGHCOMP, MEDCOMP, LOWCOMP, and MAXINTEGRITY. Using the MAXCOMPRATIO value allows the image to be stored in the smallest amount of space, but may introduce visible aberrations into the image. Using the MAXINTEGRITY value keeps the resulting image more faithful to the original, but will require more space to store.

If the CompressionQuality operator is not supplied and the destination compression format supports compression quality control, the quality defaults to MEDCOMP.

C.3 Image Processing Operators

The image processing operators supported by Visual Information Retrieval directly change the way the image looks on the display. The operators supported by Visual Information Retrieval represent only a fraction of all possible image processing operations, and the operators are not intended for users performing intricate image analysis.

C.3.1 Cut

The **Cut** operator is used to create a subset of the original image. The values supplied to the cut operator are the origin coordinates (x,y) of the cut window in the source image, and the width and height of the cut window in pixels. This operator is applied before any scaling that is requested.

If the Cut operator is not supplied, the entire source image is used.

C.3.2 Scale

The **Scale** operator enlarges or reduces the image by the ratio given as the value for the operator. If the value is greater than 1.0, then the destination image will be scaled up (enlarged). If the value is less than 1.0, then the output will be scaled down (reduced). A scale value of 1.0 has no effect, but is not an error. No scaling is applied to the source image if the Scale operator is not passed to the process( ) method.

There are two scaling techniques used by Visual Information Retrieval. The first technique is scaling by sampling and it is used only if the requested compression quality is MAXCOMPRATIO or HIGHCOMP, or if the image is being scaled up in both dimensions. This scaling technique works by selecting the source image pixel that is closest to the pixel being computed by the scaling algorithm and using the color of that pixel. This technique is faster, but results in a poorer quality image.

The second scaling technique is scaling by averaging, and it is used in all other cases. This technique works by selecting several pixels that are close to the pixel being computed by the scaling algorithm and computing the average color. This technique is slower but results in a better quality image.

If the Scale operator is not used, the default scaling value is 1.0. This operator may not be combined with other scaling operators.

C.3.3 XScale

The **XScale** operator is similar to the scale operator but only affects the width (x-dimension) of the image. The important difference between XScale and Scale is that with XScale, scaling by sampling is used whenever the image quality is specified to be MAXCOMPRATIO or HIGHCOMP, and XScale is not dependent on if the image is being scaled up or down.

This operator may be combined with the YScale operator to scale each axis differently. It may not be combined with other scaling operators (Scale, FixedScale, MaxScale).

C.3.4 YScale

The **YScale** operator is similar to the scale operator but only affects the height (y-dimension) of the image. The important difference between YScale and Scale is that with YScale, scaling by sampling is used whenever the image quality is specified to be MAXCOMPRATIO or HIGHCOMP, and YScale is not dependent on if the image is being scaled up or down.

This operator may be combined with the XScale operator to scale each axis differently. It may not be combined with other scaling operators (Scale, FixedScale, MaxScale).

C.3.5 FixedScale

The **FixedScale** operator provides an alternate method for specifying scaling values. The Scale, XScale, and YScale operators all accept floating-point scaling ratios, while the FixedScale (and MaxScale) operators specify scaling values in **pixels**. This operator is intended to simplify the creation of images with a specific size, such as thumbnail images.

The two integer values supplied to the FixedScale operator are the desired dimensions (width and height) of the destination image. The supplied dimensions may be larger or smaller (or one larger and one smaller) than the dimensions of the source image.

The scaling method used by this operator will be the same as used by the Scale operator in all cases. This operator cannot be combined with other scaling operators.

C.3.6 MaxScale

The **MaxScale** operator is a variant of the FixedScale operator that preserves the aspect ratio (relative width and height) of the source image. The MaxScale operator also accepts two integer dimensions, but these values represent the maximum value of the appropriate dimension after scaling. The final dimension may actually be less than the supplied value.

Like the FixedScale operator, this operator is also intended to simplify the creation of images with a specific size. The MaxScale operator is even better suited to thumbnail image creation than the FixedScale operator because thumbnail images created using the MaxScale operator will have the correct aspect ratios.

The MaxScale operator scales the source image to fit within the dimensions specified while preserving the aspect ratio of the source image. Because the aspect ratio is preserved, only one dimension of the destination image may actually be equal to the values supplied to the operator. The other dimension may be smaller than or equal to the supplied value. Another way to think of this scaling method is that the source image is scaled by a single scale factor that is as large as possible with the constraint that the destination image fits entirely within the dimensions specified by the MaxScale operator.

If the Cut operator is used in conjunction with the MaxScale operator, then the aspect ratio of the cut window is preserved instead of the aspect ratio of the input image.

The scaling method used by this operator is the same as used by the Scale operator in all cases. This operator cannot be combined with other scaling operators.

C.4 Format-Specific Operators

The following operators are supported only when the destination image file format is raw pixel, with the exception of the InputChannels operator, which is supported only when the source image data is raw pixel or a foreign image. It does not matter if the destination image format is set to raw pixel explicitly using the FileFormat operator, or if the raw pixel format is selected by Visual Information Retrieval automatically because the source format is raw pixel or a foreign image.

C.4.1 ChannelOrder

The **ChannelOrder** operator determines the relative order of the red, green, and blue channels (bands) within the destination raw pixel image. The order of the characters *R*, *G*, and *B* within the mnemonic value passed to this operator determines the order of these channels within the output. The header of the raw pixel image will be written such that this ordering is not lost.

For more information about the raw pixel file format and the ordering of channels in that format, see Appendix D.

C.4.2 Interleaving

The **Interleaving** operator controls the layout of the red, green, and blue channels (bands) within the destination raw pixel image. The three mnemonic values supported by this operator (BIP, BIL, and BSQ) force the output image to be band interleaved by pixel, band interleaved by line, and band sequential respectively.

For more information about the raw pixel file format, the interleaving of channels in that format, or the meaning of these interleaving values, see Appendix D.

C.4.3 PixelOrder

The **PixelOrder** operator controls the direction of pixels within a scanline in a raw pixel image. The value Normal indicates that the leftmost pixel of a scanline will

appear first in the image data stream. The value Reverse causes the rightmost pixel of the scanline to appear first.

For more information about the raw pixel file format and pixel ordering, see Appendix D.

C.4.4 ScanlineOrder

The **ScanlineOrder** operator controls the order of scanlines within a raw pixel image. The value Normal indicates that the top display scanline will appear first in the image data stream. The value Inverse causes the bottom scanline to appear first.

For more information about the raw pixel file format and scanline ordering, see Appendix D.

C.4.5 InputChannels

As stated previously, the **InputChannels** operator is supported only when the source image is in raw pixel format or if the source is a foreign image.

The InputChannels operator assigns individual bands from a multiband image to be the red, green, and blue channels for later image processing. Any band within the source image can be assigned to any channel. If desired, only a single band may be specified, and the selected band will be used as the gray channel, resulting in a grayscale output image. The first band in the image is number 1, and the band numbers passed to the Input Channels operator must be greater than or equal to 1 and less than or equal to the total number of bands in the source image. Only the bands selected by the InputChannels operator are written to the output. Other bands are not transferred, even if the output image is in raw pixel format.

It should be noted that every raw pixel or foreign image has these input channel assignments written into its header block, but that this operator overrides those default assignments.

For more information about the raw pixel file format and input channels, see Appendix D.

D

Raw Pixel Format

This appendix describes the Oracle raw pixel image format and is intended for developers and advanced users who wish to use the raw pixel format as a gateway to import unsupported image formats into Visual Information Retrieval, or who wish to use the raw pixel format as a means to directly access the pixel data in an image.

Much of this appendix is also applicable to foreign images.

D.1 Raw Pixel Introduction

Visual Information Retrieval supports many popular image formats suitable for storing artwork, photographs, and other images in an efficient, compressed way, and provides the capability to convert between these formats. However, most of these formats are proprietary to some degree, and the format of their content is often widely variable, and not particularly suited for easy access to the pixel data of the image.

The raw pixel format is useful for applications that need direct access to the pixel data without the encumbrance of the complex computations required to determine the location of pixels within a compressed data stream. This simplifies reading the image for applications that are performing pixel-oriented image processing, such as filtering and edge detection. This format is even more useful to applications that need to write data back to the image. Because changing even a single pixel in a compressed image can have implications for the entire image stream, providing an uncompressed format enables applications to write pixel data directly, and later compress the image with a single call to the process( ) method.

This format is also useful to users who have data in a format not directly supported by Visual Information Retrieval, but that is in a simple, uncompressed format. These users can simply add a raw pixel identifier and header onto their data and

import it directly into the product. For users who only need to read these images (such as for import or conversion), this capability is built into the product as foreign image support. How this capability is related to the raw pixel format is described in Section D.10.

In addition to supporting image types not already built into the product, the raw pixel format also permits the interpretation of N-band imagery, such as satellite images. Using the raw pixel format, one or three bands of an N-band image may be selected during conversion to another image format, allowing easy visualization within programs that do not otherwise support N-band images. Note that images written with the raw pixel format still may only have one or three bands.

The current version of the raw pixel format is 1.0. This appendix is applicable to raw pixel images of this release only, as the particulars of the format may change with other releases.

D.2 Raw Pixel Image Structure

A raw pixel image consists of a 4-byte image identifier, followed by a 30-byte image header, followed by an arbitrary gap of zero or more bytes, followed by pixel data.

It is worth noting that raw pixel images are never color-mapped, and therefore do not contain color look-up tables.

The raw pixel header consists of the image identifier and the image header. The image header is actually composed of several fields. Table D–1 describes the header structure.

Note that the first byte in the image is actually offset 0. All integer fields are unsigned and are stored in big endian byte order.

Table D–1 Raw Pixel Header Structure

Name	Bytes	Description
Image identifier	0:3	4-byte character array containing ASCII values for RPIX. This array identifies the image as a raw pixel image.
Image header length	4:7	Length of this header in bytes, excluding the identifier field. The value of this field may be increased to create a gap between the header fields and the pixel data in the image.

Table D–1 Raw Pixel Header Structure (Cont.)

Major version	8	Major version number of the raw pixel format used in the image.
Minor version	9	Minor version number of the raw pixel format used in the image.
Image width	10:13	Width of the image in pixels.
Image height	14:17	Height of the image in pixels.
Compression type	18	Compression type of the image: None, CCITT FAX Group 3, or CCITT FAX Group 4.
Pixel order	19	Pixel order of the image: Normal or Reverse.
Scanline order	20	Scanline order of the image: Normal or Inverse.
Interleave	21	Interleave type of the image: BIP, BIL, or BSQ.
Number of bands	22	Number of bands in the image. Must be in the range of 1 to 255.
Red channel number	23	The band number of the channel to use as a default for red. This field is the gray channel number if the image is grayscale.
Green channel number	24	The band number of the channel to use as a default for green. This field is zero if the image is grayscale.
Blue channel number	25	The band number of the channel to use as a default for blue. This field is zero if the image is grayscale.
Reserved area	26:33	Not currently used. All bytes *must* be zero.

D.3 Raw Pixel Header Field Descriptions

This section describes the fields of the raw pixel header in greater detail.

Image Identifier

Occupying the first 4 bytes of a raw pixel image, the identifier string must always be set to the ASCII values RPIX (hex 52 50 49 58). These characters identify the image as being encoded in RPIX format.

This string is currently independent of the raw pixel version.

Image Header Length

The raw pixel reader uses the value stored in this field to find the start of the pixel data section within a raw pixel image. To find the offset of the pixel data in the image, the reader adds the length of the image identifier (always 4) to the value in the image header-length field. Thus, for raw pixel 1.0 images with no post-header gap, the pixel data starts at offset 34.

For raw pixel version 1.0 images, this field normally contains the integer value 30, which is the length of the raw pixel image header (not including the image identifier). However, the raw pixel format allows this field to contain any value equal to or greater than 30. Any information in the space between the end of the header data and the start of the pixel data specified by this header length is ignored by the raw pixel reader. This is useful for users who wish to add a raw pixel header onto an existing image whose pixel data area is compatible with the raw pixel format. In this case, the header length would be set to 30 plus the length of the existing header. The maximum length of this header is 4294967265 bytes (the maximum value that can be stored in the 4-byte unsigned field minus the 30-byte header required by the raw pixel image format).

This field is stored in big endian byte order.

Major Version

A single-byte integer containing the major version number of the raw pixel format version used to encode the image. The current raw pixel version is 1.0, therefore this field is 1.

Minor Version

A single-byte integer containing the minor version number of the raw pixel format version used to encode the image. The current raw pixel version is 1.0, therefore this field is 0.

Image Width

The width (x-dimension) of the image in pixels.

Although this field is capable of storing an image dimension in excess of 4 billion pixels, limitations within the product require that this field be in the range 1 <= width <= 32767.

This field is stored in big endian byte order.

Image Height

The height (y-dimension) of the image in pixels.

Although this field is capable of storing an image dimension in excess of 4 billion pixels, limitations within this product require that this field be in the range 1 <= height <= 32767.

This field is stored in big endian byte order.

Compression Type

This field contains the compression type of the raw pixel image. As of version 1.0, this field can contain the following values:

Value	Name	Compression
1	NONE	No compression
2	FAX3	CCITT Group 3 compression
3	FAX4	CCITT Group 4 compression

For grayscale, RGB, and N-band images, the image is always uncompressed, and only a value of 0 is valid. If the compression type is value 1 or 2, then the image is presumed to be monochrome. In this case, the image is presumed to contain only a single band, and must specify NORMAL pixel order, NORMAL scanline order, and BIP interleave.

Pixel Order

This field describes the pixel order within the raw pixel image. Normally, pixels in a scanline are ordered from left to right, along the traditional positive x-axis. However, some applications require that scanlines be ordered from right to left.

This field can contain the following values:

Value	Name	Pixel Order
1	NORMAL	Left-most pixel first
2	REVERSE	Right-most pixel first

This field cannot contain 0, as this indicates an unspecified pixel order; if it does contain 0, the image cannot be interpreted. For images with FAX3 and FAX4 compression types, this field must contain the value 1.

Scanline Order

This field describes the scanline order within the raw pixel image. Normally, scanlines in an image are ordered from top to bottom. However, some applications require that scanlines are ordered from bottom to top.

This field can contain the following values:

Value	Name	Scanline Order
1	NORMAL	Topmost scanline first
2	INVERSE	Bottommost scanline first

This field cannot contain 0, as this indicates an unspecified scanline order; if it does contain 0, the image cannot be interpreted. For images with FAX3 and FAX4 compression types, this field must contain the value 1.

Interleaving

This field describes the interleaving of the various bands within a raw pixel image. For more information on the meaning of the various interleave options, see Section D.5.3.

This field can contain the following values:

Value	Name	Interleave
1	BIP	Band interleave by pixel, also known as chunky
2	BIL	Band interleave by line
3	BSQ	Band sequential, also known as planar

This field cannot contain 0, as this indicates an unspecified interleave; if it does contain 0, the image cannot be interpreted. For images with FAX3 and FAX4 compression types, this field must contain the value 1.

Number of Bands

This field contains the number of bands or planes in the image, and must be in the range 1 <= number of bands <= 255. This field cannot contain the value 0.

For CCITT images, this field must contain the value 1.

Red Channel Number

This field contains the number of the band that is to be used as the red channel during image conversion operations. This may be used to change the interpretation of a normal RGB image, or to specify a default band to be used as red in an N-band image. This default may be overridden using the inputChannels operator in the process( ) or processCopy( ) methods.

If the image has only one band, or only one band from an N-band image should be selected for display, then the band number should be encoded as the red channel. In this case, the green and blue channels should be set to 0.

This field cannot contain the value 0. Only values in the range 1 <= red <= number-of-bands may be specified.

Green Channel Number

This field contains the number of the band that is to be used as the green channel during image conversion operations. This may be used to change the interpretation of a normal RGB image, or to specify a default band to be used as green in an N-band image. This default may be overridden using the inputChannels operator in the process( ) or processCopy( ) methods.

If the image has only one band, or only one band from an N-band image should be selected for display, then the band number should be encoded as the red channel. In this case, the green and blue channels should be set to 0.

This field can contain values in the range 1 <= green <= number-of-bands.

Blue Channel Number

This field contains the number of the band that is to be used as the blue channel during image conversion operations. This may be used to change the interpretation of a normal RGB image, or to specify a default band to be used as blue in an N-band image. This default may be overridden using the inputChannels operator in the process() or processCopy() command.

If the image has only one band, or only one band from an N-band image should be selected for display, then the band number should be encoded as the red channel. In this case, the green and blue channels should be set to 0.

This field can contain values in the range 1 <= blue <= number-of-bands.

Reserved Area

The application of these 8 bytes is currently under development, but they are reserved even within raw pixel version 1.0 images. These bytes must all be cleared to zero. Failure to do so will create undefined results.

D.4 Raw Pixel Post-Header Gap

Apart from the image identifier and the image header, raw pixel version 1.0 images contain an optional post-header "gap", which precedes the actual pixel data. Unlike the reserved area part of the image header, the bytes in this gap can contain any values the user wishes. This is useful to store additional metadata about the image, which in some cases may be the actual image header from another file format.

However, because there is no standard for the information stored in this gap, care must be taken if metadata is stored in this area as other users may interpret this data differently. It is also worth noting that when a raw pixel image is processed, information stored in this gap is not copied to the destination image. In the case of the process( ) method, which writes its output to the same location as the input, the source information will be lost unless the transaction in which the processing took place is rolled back.

D.5 Raw Pixel Data Section and Pixel Data Format

The data section of a raw pixel image is where the actual pixel data of an image is stored; this area is sometimes called the *bitmap data*. This section describes the layout of the bitmap data.

For images using CCITT compression, the bitmap data area stores the raw CCITT stream with no additional header. The rest of this section applies only to uncompressed images.

Bitmap data in a raw pixel image is stored as 8-bit per plane, per-pixel, direct-color packed data. There is no pixel, scanline, or band blocking or padding. Scanlines may be presented in the image as either topmost first or bottommost first. Within a scanline, pixels may be ordered left-most first, or right-most first. All these options are affected by the interleave field in a relatively straightforward way. Examples are provided inthe following sections.

D.5.1 Scanline Ordering

On the display screen, an image may look similar to the following:

1111111111...
2222222222...
3333333333...
4444444444...

Each digit represents a single pixel; the value of the digit is the scanline that the pixel is on.

Generally, the scanline that forms the upper or top-most row of pixels is stored in the image data stream before lower scanlines. The image would appear as follows in the bitmap data stream:

...1111111111...2222222222...3333333333...4444444444...

Note that the first scanline appears earlier than the remaining scanlines. The raw pixel format refers to this scanline ordering as NORMAL.

However, some applications prefer that the bottom-most scanline appear in the data stream first:

...4444444444...3333333333...2222222222...1111111111...

The raw pixel format refers to this scanline ordering as INVERSE.

D.5.2 Pixel Ordering

On the screen, a scanline of an image may look like the following:

...123456789...
Each digit represents a single pixel; the value of the digit is the column that the pixel is on.

Generally, the data that forms the left-most pixels is stored in the image data stream before pixels toward the right. The scanline would appear as follows in the bitmap data stream:

...123456789...
Note that the left pixel appears earlier than the remaining pixels. The raw pixel format refers to this pixel ordering as NORMAL.

However, some applications prefer that the right-most pixel appear in the data stream first:

...987654321...
The raw pixel format refers to this pixel ordering as REVERSE.

D.5.3 Band Interleaving

Band interleaving describes the relative location of different bands of pixel data within the image buffer.

Note that for the purposes of this project, bands are ordered by their appearance in an image data stream, with 1 being the first band and n being the last band. Band 0 would indicate no band or no data.

Band Interleaved by Pixel (BIP), or Chunky

BIP, or chunky, images place the various bands or channels of pixel data sequentially by pixel, so that all data for one pixel is in one place. If the bands of the image are the red, green, and blue channels, then a BIP image might look similar to this:

```
scanline 1: RGBRGBRGBRGBRGBRGBRGB...
scanline 2: RGBRGBRGBRGBRGBRGBRGB...
scanline 3: RGBRGBRGBRGBRGBRGBRGB...
...
```

Band Interleaved by Line (BIL)

BIL images place the various bands of pixel data sequentially by scanline, so that data for one pixel is spread across multiple notional “rows” of the image. This reflects the data organization of a sensor that buffers data by scanline. If the bands of the image are the red, green, and blue channels, then a BIL image might look similar to this:

```
scanline 1: RRRRRRRRRRRRRRRRRRRR...
            GGGGGGGGGGGGGGGGGGGG...
            BBBBBBBBBBBBBBBBBBBB...
scanline 2: RRRRRRRRRRRRRRRRRRRR...
            GGGGGGGGGGGGGGGGGGGG...
            BBBBBBBBBBBBBBBBBBBB...
scanline 3: RRRRRRRRRRRRRRRRRRRR...
            GGGGGGGGGGGGGGGGGGGG...
            BBBBBBBBBBBBBBBBBBBB...
...
```

Band Sequential (BSQ), or Planar

Planar images place the various bands of pixel data sequentially by bit-plane, so that data for one pixel is spread across multiple “planes” of the image. This reflects the data organization of some video buffer systems, which control the different electron guns of a display from different locations in memory. If the bands of the

image are the red, green, and blue channels, then a planar image might look similar to this:

```
plane 1: RRRRRRRRRRRRRRRRR…  (part of scanline 1)
         RRRRRRRRRRRRRRRRR…  (part of scanline 2)
         RRRRRRRRRRRRRRRRR…  (part of scanline 3)
…
plane 2: GGGGGGGGGGGGGGGGG…  (part of scanline 1)
         GGGGGGGGGGGGGGGGG…  (part of scanline 2)
         GGGGGGGGGGGGGGGGG…  (part of scanline 3)
…
plane 3: BBBBBBBBBBBBBBBBB…  (part of scanline 1)
         BBBBBBBBBBBBBBBBB…  (part of scanline 2)
         BBBBBBBBBBBBBBBBB…  (part of scanline 3)
…
```

D.5.4 N-Band Data

The raw pixel format supports up to 255 bands of data in an image. The relative location of these bands of data in the image is described in Section D.5.3, which gives examples of interleaving for 3 bands of data.

In the case of a single band of data, there is no interleaving; all three schemes are equivalent. Examples of interleaving other numbers of bands are given in the following table. All images have three scanlines and four columns. Each band of each pixel is represented by a single-digit band number. Numbers in *italic* represent the second scanline of the image, and numbers in **boldface** represent the third scanline of the image.

Bands	BIP	BIL	BSQ
2	12121212 *12121212* **12121212**	11112222 *11112222* **11112222**	1111*1111***1111** 2222222**2222**
4	1234123412341234 *1234123412341234* **1234123412341234**	1111222233334444 *1111222233334444* **1111222233334444**	1111*1111***1111** 2222222**2222** 3333333**3333** 4444444**4444**
5	12345123451234512345 *12345123451234512345* **12345123451234512345**	11112222333344445555 *11112222333344445555* **11112222333344445555**	1111*1111***1111** 2222222**2222** 3333333**3333** 4444444**4444** 5555555**5555**

D.6 Raw Pixel Header for C Language Structure

The following C language structure describes the raw pixel header in a programmatic way. This structure is stored unaligned in the image file (fields are aligned on 1-byte boundaries), and all integers are stored in big endian byte order.

```
struct RawPixelHeader
{
unsigned char identifier[4];/* Always "RPIX" */

unsigned long hdrlength; /* Length of this header in bytes */
/* Including the hdrlength field */
/* Not including the identifier field */
/* &k.hdrlength + k.hdrlength = pixels */

unsigned char majorversion; /* Major revision # of RPIX format */
unsigned char minorversion; /* Minor revision # of RPIX format */

unsigned long width; /* Image width in pixels */
unsigned long height; /* Image height in pixels */

unsigned char comptype; /* Compression (none, FAXG3, FAXG4, ... ) */
unsigned char pixelorder; /* Pixel order */
unsigned char scnlorder; /* Scanline order */
unsigned char interleave; /* Interleaving (BIP/BIL/Planar) */

unsigned char numbands; /* Number of bands in image (1-255) */
unsigned char rchannel; /* Default red channel assignment */
unsigned char gchannel; /* Default green channel assignment */
unsigned char bchannel; /* Default blue channel assignment */
/* Grayscale images are encoded in R */
/* The first band is '1', not '0' */
/* A value of '0' means "no band" */

unsigned charreserved[8];/* For later use */
};
```

D.7 Raw Pixel Header for C Language Constants

The following C language constants define the values used in the raw pixel header.

```
#define RPIX_IDENTIFIER"RPIX"

#define RPIX_HEADERLENGTH              30
```

```
#define RPIX_MAJOR_VERSION             1
#define RPIX_MINOR_VERSION0

#define RPIX_COMPRESSION_UNDEFINED     0
#define RPIX_COMPRESSION_NONE          1
#define RPIX_COMPRESSION_CCITT_FAX_G3  2
#define RPIX_COMPRESSION_CCITT_FAX_G4  3
#define RPIX_COMPRESSION_DEFAULT       RPIX_COMPRESSION_NONE

#define RPIX_PIXEL_ORDER_UNDEFINED     0
#define RPIX_PIXEL_ORDER_NORMAL        1
#define RPIX_PIXEL_ORDER_REVERSE       2
#define RPIX_PIXEL_ORDER_DEFAULT       RPIX_PIXEL_ORDER_NORMAL

#define RPIX_SCANLINE_ORDER_UNDEFINED  0
#define RPIX_SCANLINE_ORDER_NORMAL     1
#define RPIX_SCANLINE_ORDER_INVERSE    2
#define RPIX_SCANLINE_ORDER_DEFAULT    RPIX_SCANLINE_ORDER_NORMAL

#define RPIX_INTERLEAVING_UNDEFINED    0
#define RPIX_INTERLEAVING_BIP          1
#define RPIX_INTERLEAVING_BIL          2
#define RPIX_INTERLEAVING_BSQ          3
#define RPIX_INTERLEAVING_DEFAULT      RPIX_INTERLEAVING_BIP

#define RPIX_CHANNEL_UNDEFINED         0
```

Note that the various macros for the undefined values are meant to be illustrative and not necessarily used, except for RPIX_CHANNEL_UNDEFINED, which is used for the green and blue channels of single band images.

D.8 Raw Pixel PL/SQL Constants

The following PL/SQL constants define the values used in the raw pixel information:

```
CREATE OR REPLACE PACKAGE ORDVirConstants AS
  RPIX_HEADER_LENGTH_1_0   CONSTANT INTEGER := 34;
END ORDVirConstants;
```

D.9 Raw Pixel Images Using CCITT Compression

Although the raw pixel format is generally aimed at uncompressed, direct-color images, provision is also made to store monochrome images using CCITT Fax Group 3 or Fax Group 4 compression. This is useful for storing scans of black and white pages, such as for document management applications. These images are generally infeasible to store even as merely grayscale, as the unused data bits combined with the very high resolution used in these images would use excessive disk space.

Raw pixel images using CCITT compression are treated as normal raw pixel images, with the following restrictions:

- The compression type field must contain the value 1 or 2, as outlined in Section D.3 (FAX3 or FAX4).
- The pixel order field must contain the value 1 (normal pixel order).
- The scanline order field must contain the value 1 (normal scanline order).
- The interleave field must contain the value 1 (BIP interleave).
- The number of bands field must contain the value 1 (one band).
- The red channel number field must contain the value 1.
- The green channel number and blue channel number fields must contain the value 0 (no band).

In addition to these restrictions, applications that attempt to access pixel data directly will need to understand how to read and write the CCITT formatted data.

D.10 Foreign Image Support and the Raw Pixel Format

Visual Information Retrieval provides support for reading certain foreign images that can be described in terms of a few simple parameters, and whose data is arranged in a certain straightforward way within the image file. There is no list of the supported formats because the list would be so large and continually changing. Instead, there are some simple guidelines to determine if an image can be read using the foreign image support in the product. These rules are summarized in the following sections.

Header

Foreign images may have any header, in any format, as long as its length does not exceed 4294967265 bytes. As has been noted before, all information in this header will be ignored.

Image Width

Foreign images may be up to 32767 pixels wide.

Image Height

Foreign images may be up to 32767 pixels high.

Compression Type

Foreign images must be uncompressed or compressed using CCITT Fax Group 3 or Fax Group 4. Other compression schemes, such as run-length encoding, are not currently supported.

Pixel Order

Foreign images may store pixels from left to right or right to left. Other pixel ordering schemes are not currently supported.

Scanline Order

Foreign images may have top-first or bottom-first scanline orders. Scanlines that are adjacent in the image display must be adjacent in the image storage. Some image formats stagger their image scanlines so that, for example, scanlines 1, 5, 9, and so forth, are adjacent, and then 2, 6, 10 are also adjacent. This is not currently allowed.

Interleaving

Foreign images must use BIP, BIL, or BSQ interleaving. Other arrangements of data bands are not allowed, nor may bands have any pixel, scanline, or band-level blocking or padding.

Number of Bands

Foreign images may have up to 255 bands of data. If there are more bands of data, the first 255 can be accessed *if* the interleaving of the image is band sequential. In this case, the additional bands of data lie past the accessible bands and do not affect the layout of the first 255 bands. Images with other interleaving types may not have more than 255 bands because the additional bands will change the layout of the bitmap data.

Trailer

Foreign images may have an image trailer following the bitmap data, and this trailer may be of arbitrary length. However, such data is completely ignored by the product, and there is no method (or need) to specify the presence of length of such a trailer.

If an image with such a trailer is modified with the process( ) or processCopy( ) methods, the resulting image will not contain this trailer. In the case of the processCopy method, the source image will still be intact.

E

Deprecated Features

Since release 8.1.5, the ORDVirB and ORDVirF object types have been replaced by ORDVir. These object types will be supported in release 8.1.7, but will no longer exist in the next release (after release 8.1.7).

The new ORDVir type is based on the ORDImage type in the Oracle *inter*Media product. The old interface and object types have been replaced by the interface described in Chapter 4.

This appendix describes the old interface, for those users who still need to access legacy data. Please see the migration functions (migrateFromORDVirB( ) and migrateFromORDVirF( ) methods) in Chapter 4 for information on moving to the new object type.

The old interface will be removed in the next release after release 8.1.7.

The obsolete Visual Information Retrieval library consists of:

- Object types -- See Section E.1.
- Methods -- See Section E.2.
- Operators -- See Section E.3.

The examples in this chapter assume that the stockphotos table has been created and filled with some photographic images. The table was created using the following SQL statement:

```
CREATE TABLE stockphotos (photo_id NUMBER, photographer (VARCHAR2(64),
            annotation (VARCHAR2(255), photo ORDSYS.ORDVIRB);
```

When you are storing or copying images, you must first create an empty BLOB in the table. Use the empty_blob( ) function, which has the following format:

```
ORDSYS.ORDVIRB(empty_blob(),NULL,NULL,NULL,NULL,NULL,NULL,NULL)
```

E.1 Object Types

The obsolete Visual Information Retrieval object types are as follows:

- ORDVirB -- Supports images stored in an Oracle8*i* BLOB
- ORDVirF -- Supports images stored in an Oracle8*i* external file (BFILE)

This section presents reference information on the object types.

For more information on BLOBs and BFILEs, see *Oracle8i Application Developers Guide: Large Objects (LOBs).*

ORDVirB Object Type

The ORDVirB object type supports storage and retrieval of image data in a BLOB within an Oracle database. This object type is defined as follows:

```
CREATE TYPE ORDVIRB
(
-- TYPE ATTRIBUTES
content             BLOB,
height              INTEGER,
width               INTEGER,
contentLength       INTEGER,
fileFormat          VARCHAR2(64),
contentFormat       VARCHAR2(64),
compressionFormat   VARCHAR2(64),
signature           RAW(2000),
-- METHOD DECLARATION
    MEMBER PROCEDURE copyContent(dest IN OUT NOCOPY BLOB),
  MEMBER PROCEDURE setProperties(SELF IN OUT ORDVIRB),
  MEMBER PROCEDURE setProperties(SELF IN OUT ORDVIRB, description IN VARCHAR2),
  MEMBER PROCEDURE process    (SELF    IN OUT ORDVIRB,
                               command  IN     VARCHAR2)
  MEMBER PROCEDURE processCopy(command  IN     VARCHAR2,
                               dest     IN OUT NOCOPY BLOB),
  MEMBER FUNCTION  getMimeType RETURN VARCHAR2,
  MEMBER FUNCTION  getContent  RETURN BLOB,
  MEMBER FUNCTION  getContentLength RETURN INTEGER,
  MEMBER PROCEDURE deleteContent (SELF IN OUT ORDVirB),
  MEMBER PROCEDURE Analyze (SELF IN OUT ORDVIRB),
  MEMBER FUNCTION  getHeight   RETURN INTEGER,
  MEMBER FUNCTION  getWidth    RETURN INTEGER,
  MEMBER FUNCTION  getFileFormat RETURN VARCHAR2,
  MEMBER FUNCTION  getContentFormat RETURN VARCHAR2,
  MEMBER FUNCTION  getCompressionFormat RETURN VARCHAR2,
  MEMBER FUNCTION  getSignature RETURN RAW,
  MEMBER FUNCTION  checkProperties RETURN BOOLEAN
);
```

Where the attributes are:

- 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 or JFIF)
- contentFormat: Type of image (such as, monochrome or 8-bit grayscale)
- compressionFormat: Compression type of image
- signature: Feature vector (signature) for content-based retrieval

In PL/SQL, data is moved with the DBMS LOB package. From the client, data is moved using OCI LOB calls. The ORDVirB object type does not supply routines for moving data piece by piece.

ORDVirF Object Type

The ORDVirF object type supports storage and retrieval of image data in external files (BFILEs, which are not stored in the database). BFILEs are assumed to be read-only, and this is reflected in the member procedures. This object type is defined as follows:

```
CREATE TYPE ORDVIRF
(
  -- TYPE ATTRIBUTES
content             BFILE,
height              INTEGER,
width               INTEGER,
contentLength       INTEGER,
fileFormat          VARCHAR2(64),
contentFormat       VARCHAR2(64),
compressionFormat   VARCHAR2(64),
signature           RAW(2000),
-- METHOD DECLARATION
  MEMBER PROCEDURE copyContent(dest IN OUT NOCOPY BLOB),
  MEMBER PROCEDURE setProperties(SELF IN OUT ORDVIRF),
  MEMBER PROCEDURE setProperties(SELF IN OUT ORDVIRF, description IN VARCHAR2),
  MEMBER PROCEDURE processCopy(command IN VARCHAR2,
                               dest    IN OUT NOCOPY BLOB),
  MEMBER FUNCTION  getMimeType RETURN VARCHAR2,
  MEMBER FUNCTION  getContent  RETURN BFILE,
  MEMBER FUNCTION  getContentLength RETURN INTEGER,
  MEMBER PROCEDURE Analyze (SELF IN OUT ORDVIRF),
  MEMBER FUNCTION  getHeight   RETURN INTEGER,
  MEMBER FUNCTION  getWidth    RETURN INTEGER,
  MEMBER FUNCTION  getFileFormat RETURN VARCHAR2,
  MEMBER FUNCTION  getContentFormat RETURN VARCHAR2,
  MEMBER FUNCTION  getCompressionFormat RETURN VARCHAR2,
  MEMBER FUNCTION  getSignature RETURN RAW,
  MEMBER FUNCTION  checkProperties RETURN BOOLEAN
);
```

Where the attributes are:

- 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 or JFIF)
- contentFormat: Type of image (such as, monochrome or 8-bit grayscale)
- compressionFormat: Compression type of image
- signature: Feature vector (signature) for content-based retrieval

E.2 Methods

This section presents reference information on the methods used for image manipulation.

Visual Information Retrieval methods are as follows:

- analyze( ): Creates the signature of an image based on the image characteristics.
- checkProperties( ): Verifies that the characteristics stored as image attributes match the properties of the actual image.
- copyContent( ): Copies only the image portion of a BLOB or BFILE to a BLOB.
- deleteContent( ): Deletes the contents of an image stored in a BLOB.
- getCompressionFormat( ): Returns the compression type used on an image.
- getContent( ): Returns the BLOB or BFILE containing an image.
- getContentFormat( ): Returns the type of the image.
- getContentLength( ): Returns the size of the image in bytes.
- getFileFormat( ): Returns the file type of an image.
- getHeight( ): Returns the height of the image in pixels.
- getMimeType( ): Returns the MIME type of an image.
- getSignature( ): Returns the signature of an image.
- getWidth( ): Returns the width of an image in pixels.
- process( ): Processes an image in place (for example, modifies it or converts it to another format); available for BLOBs only.
- processCopy( ): Copies an image and processes the copy (for example, modifies it or converts it to another format); available for BLOBs and BFILEs, but the destination must be a BLOB.
- setProperties( ): Obtains and stores the attributes of an image.

For more information on object types and methods, see *Oracle8i Concepts*.

analyze( ) Method

Format

Analyze( );

Description

Analyzes an image BLOB or BFILE, derives information relating to the visual attributes (including a score for each), and creates the image signature.

Usage

The Analyze( ) method creates the image attribute signature, which is necessary for any content-based retrieval. Whenever you are working with a new or changed image, you should also use the SetProperties( ) method to set the other image characteristics.

Signatures for facial images can be created using an optional third-party software package from Viisage Technology, Inc. After creating a facial signature, Visual Information Retrieval can convert the signature to a standard format and then compare the signatures using the Score( ) and Similar( ) operators.

The Analyze( ) method is functionally equivalent to the Analyze( ) operator. You must use the Analyze( ) method when working with foreign images.

Examples

Create the signatures for all images in the stockphotos table.

```
DECLARE
   temp_image   ORDSYS.ORDVirB;
   temp_id      INTEGER;
   cursor c1 is select id, image from stockphotos for update;
BEGIN
   OPEN c1;
   LOOP
      fetch c1 into temp_id, temp_image;
      EXIT WHEN c1%NOTFOUND;
```

```
 -- Generate signature and set the properties for the image.
      temp_image.analyze;
      temp_image.setProperties;
      UPDATE stockphotos SET photo = temp_image WHERE photo_id = temp_id;
   END LOOP;
   CLOSE c1;
END;
```

checkProperties Method

Format

checkProperties RETURN BOOLEAN;

Description

Verifies that the properties stored in attributes of the image object match the properties of the image stored in the BLOB or BFILE. This method should not be used for foreign images.

Parameter

None.

Returns

BOOLEAN

Usage

Use this method to verify that the image attributes match the actual image.

Examples

Check the image attributes.

```
virb1            ORDSYS.ORDVirB;
properties_match BOOLEAN;

...
properties_match := virb1.checkProperties;
```

copyContent( ) Method

Format

copyContent (dest IN OUT NOCOPY BLOB);

Description

Copies an image without changing it.

Parameter

dest
The destination of the new image.

Usage

This method copies the image data into the supplied BLOB.

Examples

Create a copy of the image in image1 into myblob.

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

deleteContent Method

Format

deleteContent;

Description

Deletes the contents of the image.

Parameter

None.

Usage

Use this method to delete the contents of the image BLOB. This procedure works only with BLOBs, not BFILEs.

Examples

Delete the content of an image.

```
virb1  ORDSYS.ORDVirB;

...
virb1.deleteContent;
```

getCompressionFormat Method

Format

getCompressionFormat RETURN VARCHAR2;

Description

Returns the compression type of an image. This method does not actually read the LOB; it is a simple access method that returns the value of the compressionFormat attribute.

Parameter

None.

Returns

VARCHAR2

Usage

Use this method rather than accessing the compressionFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDVirB or ORDVirF object.

Examples

Get the compression type of an image.

```
virb1             ORDSYS.ORDVirB;
compressionFormat VARCHAR2(64);

...
compressionFormat := virb1.getCompressionFormat;
```

getContent Method

Format

getContent RETURN BLOB;

getContent RETURN BFILE;

Description

Returns the LOB locator of the BLOB or BFILE containing the image. This is a simple access method that returns the value of the content attribute.

Parameter

None.

Returns

BLOB or BFILE, depending on how the image is stored.

Usage

Use this method rather than accessing the content attribute directly to protect yourself from potential changes to the internal representation of the ORDVirB or ORDVirF object.

Examples

Get the LOB locator for an image.

```
virb1   ORDSYS.ORDVirB;
content BLOB;
...
content := virb1.getContent;
```

getContentFormat Method

Format

getContentFormat RETURN VARCHAR2;

Description

Returns the type of an image (such as monochrome or 8-bit grayscale). This method does not actually read the LOB; it is a simple access method that returns the value of the contentFormat attribute.

Parameter

None.

Returns

VARCHAR2

Usage

Use this method rather than accessing the contentFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDVirB or ORDVirF object.

Examples

Get the type of an image.

```
virb1         ORDSYS.ORDVirB;
contentFormat VARCHAR2(64);

...
contentFormat := virb1.getContentFormat;
```

getContentLength Method

Format

getContentLength RETURN INTEGER;

Description

Returns the size of the on-disk image in bytes. This method does not actually read the LOB; it is a simple access method that returns the value of the contentLength attribute.

Parameter

None.

Returns

INTEGER

Usage

Use this method rather than accessing the contentLength attribute directly to protect yourself from potential changes to the internal representation of the ORDVirB or ORDVirF object.

Examples

Get the content length of an image.

```
virb1         ORDSYS.ORDVirB;
contentLength INTEGER;

...
contentLength := virb1.getContentLength;
```

getFileFormat Method

Format

getFileFormat RETURN VARCHAR2

Description

Returns the file type of an image (such as TIFF or JFIF). This method does not actually read the LOB; it is a simple access method that returns the value of the fileFormat attribute.

Parameter

None.

Returns

VARCHAR2

Usage

Use this method rather than accessing the fileFormat attribute directly to protect yourself from potential changes to the internal representation of the ORDVirB or ORDVirF object.

Examples

Get the file type of an image.

```
virb1      ORDSYS.ORDVirB;
fileFormat VARCHAR2(64);

...
fileFormat := virb1.getFileFormat;
```

getHeight Method

Format

getHeight RETURN INTEGER;

Description

Returns the height of an image in pixels. This method does not actually read the LOB; it is a simple access method that returns the value of the height attribute.

Parameter

None.

Returns

INTEGER

Usage

Use this method rather than accessing the height attribute directly to protect yourself from potential changes to the internal representation of the ORDVirB or ORDVirF object.

Examples

Get the height of an image.

```
virb1  ORDSYS.ORDVirB;
height INTEGER;

...
height := virb1.getHeight;
```

getMimeType Method

Format

getMimeType RETURN VARCHAR2;

Description

Returns the Multipurpose Internet Mail Extension (MIME) type of an image (such as image/jpeg or image/tiff). This method returns the MIME type based on the fileFormat of the image. See Appendix A for the MIME type associated with each supported file format.

Parameter

None.

Returns

VARCHAR2

Usage

Use this method to obtain the MIME type of the image. The MIME type is required by Web browsers along with the image content. It tells the Web browser how to interpret the image content.

Examples

Get the MIME type of an image.

```
virb1      ORDSYS.ORDVirB;
mimeType   VARCHAR2(64);

...
mimeType := virb1.getMimeType;
```

getSignature Method

Format

getSignature RETURN RAW;

Description

Returns the signature of an image. This method does not actually create the image signature; it is a simple access method that returns the contents of the signature attribute.

Parameter

None.

Returns

RAW

Usage

Use this method rather than accessing the signature attribute directly to protect yourself from potential changes to the internal representation of the ORDVirB or ORDVirF object.

Examples

Get the signature of an image.

```
virb1     ORDSYS.ORDVirB;
 signature RAW(2000);

...
signature := virb1.getSignature;
```

getWidth Method

Format

getWidth RETURN INTEGER;

Description

Returns the width of an image in pixels. This method does not actually read the LOB; it is a simple access method that returns the value of the width attribute.

Parameter

None.

Returns

INTEGER

Usage

Use this method rather than accessing the width attribute directly to protect yourself from potential changes to the internal representation of the ORDVirB or ORDVirF object.

Examples

Get the width of an image.

```
virb1  ORDSYS.ORDVirB;
width  INTEGER;

...
width := virb1.getWidth;
```

process( ) Method

Format

process (command IN VARCHAR2);

Description

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

Parameter

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

Usage

You can change one or more of the image attributes shown in Table E–1. Table E–2 shows additional changes can be made to raw pixel and foreign images. See Appendix A for information on the supported format combinations. See Appendix C for a more complete description of each operator.

Table E–1 Image Processing Operators

Parameter Name	Usage	Values
compressionFormat	Compression type / format	JPEG, SUNRLE, BMPRLE, TARGARLE, LZW, LZWHDIFF, FAX3, FAX4, HUFFMAN3, Packbits, GIFLZW
compressionQuality	Compression quality	MAXCOMPRATIO, MAXINTEGRITY, LOWCOMP, MEDCOMP, HIGHCOMP
contentFormat	Image type / pixel / data format	MONOCHROME, 8BITGRAYSCALE, 8BITGREYSCALE, 8BITLUT, 24BITRGB
cut	Window to cut (origin.x origin.y width height)	(INTEGER INTEGER INTEGER INTEGER) Maximum value is 65535.
fileFormat	File format of the image	BMPF, CALS, GIFF, JFIF, PICT, RASF, RPIX, TGAF, TIFF
fixedScale	Scale to a specific size in pixels (width, height)	(INTEGER INTEGER)

Table E–1 Image Processing Operators (Cont.)

Parameter Name	Usage	Values
maxScale	Scale to a specific size in pixels, while maintaining the aspect ratio (maxWidth, maxHeight)	(INTEGER INTEGER)
scale	Scale factor (for example, 0.5 or 2.0); preserves aspect ratio	<FLOAT> positive
xScale	X-axis scale factor (Default is 1.)	<FLOAT> positive
yScale	Y-axis scale factor (Default is 1.)	<FLOAT> positive

Table E–2 Additional Image Processing Operators for Raw Pixel and Foreign Images

Parameter Name	Usage	Values
ChannelOrder	Indicates the relative position of the red, green, and blue channels (bands) within the image.	RGB (default), RBG, GRB, GBR, BRG, BGR
InputChannels	For multiband images, specify either one (grayscale) or three integers indicating which channels to assign to red (first), green (second), and blue (third). Note that this parameter affects the source image, not the destination.	INTEGER or INTEGER INTEGER INTEGER
Interleave	Controls band layout within the image: Band interleaved by pixel Band interleaved by line Band sequential	BIP (default), BIL, BSQ
PixelOrder	If NORMAL, then the leftmost pixel appears first in the image.	NORMAL (default), REVERSE
ScanlineOrder	If NORMAL, then the top scanline appears first in the image.	NORMAL (default), INVERSE

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

Example 1: Change the file format of image1 to GIF.

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

Example 2: Change image1 to use lower quality JPEG compression and double the size of the image, preserving the aspect ratio.

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

Note that changing the length on only one axis (for example, xScale=2.0) does not affect the length on the other axis, and would result in image distortion. Also, only the xScale and yScale parameters can be combined in a single operation. Any other combinations result in an error.

Example 3: The maxScale and fixedScale parameters are especially useful for creating thumbnail images from various-sized originals. The following line of code creates a 32-by-32 pixel thumbnail image, preserving the original aspect ratio:

```
image1.process('maxScale=32 32');
```

processCopy( ) Method

Format

processCopy (command IN VARCHAR2,

dest IN OUT NOCOPY BLOB);

Description

Copies an image BLOB or BFILE to another BLOB.

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 E–1 and Table E–2.

When using temporary LOBs, you cannot specify the same temporary LOB as both the source and the destination.

Examples

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

```
create or replace procedure copyit is
 virB1     ORDSYS.ORDVIRF;
 virB4     ORDSYS.ORDVIRB;
 mycommand   VARCHAR2(400);
begin
  select col2 into virB1 from ordvirtab  where col1 = 1;
  select col2 into virB4 from ordvirtab  where col1 = 4 for update;
  mycommand:= 'fileFormat=tiff compressionFormat = packbits
  contentFormat = 8bitlut';
  virB1.processcopy(mycommand,virB4.content);
  virB4.setproperties;
  update ordvirtab set col2 = virB4 where col1 = 4;
```

```
end;
```

setProperties( ) Method

Format

setProperties( );

Description

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

Parameters

None.

Usage

After you copied, stored, or processed an image, call this method to set the characteristics of the new image content.

This method 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 grayscale, and so forth)
- Compression type (JPEG, LZW, and so forth)

Note that the setProperties( ) method does not create the signature required for content-based retrieval. See the Analyze( ) operator in Section E.3 for details.

Examples

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

```
virB1 ORDSYS.ORDVIRB;
.
.
.
select col2 into virB1 from ordvirtab where col1 = 1 for update;
```

```
virB1.setProperties;
dbms_output.put_line('image width = '|| virB1.width );
dbms_output.put_line('image height = '|| virB1.height );
dbms_output.put_line('image size = '|| virB1.contentLength );
dbms_output.put_line('image file type = '|| virB1.fileFormat );
dbms_output.put_line('image type = '|| virB1.contentFormat );
dbms_output.put_line('image compression = '|| virB1.compressionFormat );
-- Note: signature not meaningful as displayed output.
```

Example output:

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

setProperties( ) Method for Foreign Images

Format

setProperties(description IN VARCHAR2);

Description

Allows you to write the characteristics of a foreign image (BLOB or BFILE) into the appropriate attribute fields.

Parameter

description
The image characteristics to set for the foreign image.

Usage

After you copied, stored, or processed a foreign image, call this method to set the characteristics of the new image content. Unlike the native image types described in Appendix A, foreign images do not contain information on how to interpret the bits in the file (or contain information that the product cannot understand), and you must set them explicitly.

You can set the following image characteristics for foreign files, as shown in Table E–3.

Table E–3 Image Characteristics for Foreign Files

Field	Data Type	Description
CompressionFormat	STRING	Value must be CCITTG3, CCITTG4, or NONE (default).
DataOffset	INTEGER	The offset allows the image to have a header that the product does not try to interpret. Set the offset to ignore any potential header. The value must be a positive integer less than the LOB length. Default is zero.
DefaultChannelSelection	INTEGER	For multiband images, specify either one (grayscale) or three integers indicating which channels to assign to red (first), green (second), and blue (third).
Height	INTEGER	Height of the image in pixels. Value must be a positive integer. There is no default, and a value must be specified.

Table E–3 Image Characteristics for Foreign Files(Cont.)

Field	Data Type	Description
MIME type	STRING VARCHAR2	Value must be a MIME type search as image/gif.
Interleaving	STRING	Band layout within the image. Valid styles are: ■ BIP (default): Band interleaved by pixel ■ BIL: Band interleaved by line ■ BSQ: Band sequential
NumberOfBands	INTEGER	Value must be a positive integer less than 255 describing the number of color bands in the image. Default is 3.
PixelOrder	STRING	If NORMAL (default), the leftmost pixel appears first in the file. If REVERSE, the rightmost pixel appears first.
ScanlineOrder	STRING	If NORMAL (default), the top scanline appears first in the file. If INVERSE, the bottom scanline appears first.
UserString	STRING	A 4-character descriptive string. If used, the string is stored in the fileFormat field, appended to the file format (OTHER:). Default is blank.
Width	INTEGER	Width of the image in pixels. Value must be a positive integer. There is no default, and a value must be specified.

The values supplied to setProperties( ) are written to the existing ORDVirB and ORDVirF object attributes. The fileFormat is set to OTHER: and includes the user string, if supplied.

Note that setProperties( ) does not create the signature required for content-based retrieval. See the Analyze( ) operator in Section E.3 for details.

Examples

Select the image type, and then set the attributes using the SetProperties method.

```
virB1 ORDSYS.ORDVIRB;
select col2 into virB1 from ordvirtab where col1 = 1 for update;
virB1.setProperties('width=380 height=407 dataOffset=128 bandOrder=BIL
     userString="LSAT"');
```

E.3 Operators

Visual Information Retrieval operators are located in the ORDSYS.VIR package. The operators, which are specific to content-based retrieval, are as follows:

- Analyze( ): Produces the signature of an image.
- Convert( ): Converts the signature to either big or little endian byte order based on the architecture of the host machine. This operator can also convert a Viisage facial signature to a signature usable by the Score( ) and Similar( ) or VIRScore( ) and VIRSimliar( ) operators.
- Score( ): Compares two signatures, considers the weights for the visual attributes, and computes an overall score that is the weighted sum of the distances. For example, in Section 2.3.3, overall scores (weighted sums) of 61.5 and 24 were computed for comparisons of the same two images using two different sets of weights.
- Similar( ): Compares two signatures and determines whether or not the images match, based on the weights and threshold; returns 1 if the computed distance measure (weighted average) is less than or equal to the threshold value, otherwise returns 0.

For ease of use, you can create a local synonym for the ORDSYS.VIR package. Connect to your schema and issue the following command:

```
SVRMGR> CREATE SYNONYM VIR FOR ORDSYS.VIR;
```

After creating the synonym, you can use it in calls to the operators: VIR.Analyze( ), VIR.Score( ), and so forth. Note that you must have the default CREATE SYNONYM privilege.

This section presents reference information on the operators.

Analyze( ) Operator

Format

Analyze(image IN BLOB, signature OUT RAW);

or

Analyze(image IN BFILE, signature OUT RAW);

Description

Analyzes an image BLOB or BFILE, derives information relating to the visual attributes, and creates the image signature.

Parameters

image
The BLOB or BFILE to be analyzed.

signature
The vector to contain the signature.

Usage

The Analyze( ) operator creates the image signature (or feature vector), which is necessary for any content-based retrieval. Whenever you are working with a new or changed image, you should call Analyze( ) to generate a signature and then use the SetProperties( ) method to set the other image characteristics.

Signatures for facial images can be created using an optional third-party software package from Viisage Technology, Inc. After creating a facial signature, Visual Information Retrieval can convert the signature to a standard format and then compare the signatures using the Score( ) and Similar( ) operators.

Examples

Create the signatures for all images in the stockphotos table.

```
DECLARE
   temp_image   ORDSYS.ORDVirB;
   temp_id      INTEGER;
   cursor c1 is select id, image from stockphotos for update;
BEGIN
   OPEN c1;
   LOOP
      fetch c1 into temp_id, temp_image;
      EXIT WHEN c1%NOTFOUND;

 -- Generate signature and set the properties for the image.
      ORDSYS.VIR.Analyze(temp_image.content, temp_image.signature);
      temp_image.setProperties;
      UPDATE stockphotos SET photo = temp_image WHERE photo_id = temp_id;
   END LOOP;
   CLOSE c1;
END;
```

Convert( ) Operator

Format

Convert(signature IN OUT RAW, operation IN VARCHAR2);

Description

Converts the image signature to a format usable by the host machine.

Parameters

signature
The signature of the image, as created by the Analyze( ) operator or by Viisage software. Data type is raw(2000).

operation
The processing to be done to the image signature. The following operations are available:

Operation Keyword	Description
BYTEORDER	Converts the signature to the natural byte order of the host machine.
VIISAGE	Converts the signature from the format used for Viisage facial recognition to a signature usable by the Score( ) and Similar( ) operators.

Usage

When the operation is BYTEORDER, the signature is converted to the format of the host machine regardless of its initial state.

This procedure is useful if the database is stored on a remote system, but you want to do your processing locally. If your host machine is from Sun Microsystems, Inc., the Convert( ) operator sets the signature to the big endian byte order. On an Intel Corporation machine, the operator converts the signature to the little endian byte order. Note that the images themselves are machine-independent; only the signatures need to be converted.

When the operation is VIISAGE, the signature is converted from the format used by Viisage Technology for facial recognition to the format usable by this product for Score( ) and Similar( ) operators.

Examples

Example 1: Convert the signature of the image with `photo_id=1` to a signature that is usable by the host system.

```
DECLARE
   myimage ORDSYS.ORDVirB;
   myid      INTEGER;
BEGIN
   SELECT photo INTO myimage FROM stockphotos WHERE photo_id=1 FOR UPDATE;
   ORDSYS.VIR.Convert(myimage.signature,'BYTEORDER');
   UPDATE stockphotos SET photo=myimage WHERE photo_id=1;
END;
```

Example 2: Convert the signatures of the image with `photo_id=1` from the format used for Viisage facial image recognition software to a signature usable by the Score( ) and Similar( ) operators.

```
DECLARE
   myimage ORDSYS.ORDVirB;
   myid    INTEGER;
BEGIN
   SELECT photo INTO myimage FROM stockphotos WHERE photo_id=1 FOR UPDATE;
   ORDSYS.VIR.Convert(myimage.signature,'VIISAGE');
   UPDATE stockphotos SET photo=myimage WHERE photo_id=1;
END;
```

Score( ) Operator

Format

```
Score(signature1   IN RAW,
      signature2   IN RAW,
      weightstring IN VARCHAR2);
```

where weightstring is: 'globalcolor="val" localcolor="val" texture="val" structure="val"'

or: 'facial=1'

Description

Compares the signatures of two images and returns a number representing the weighted sum of the distances for the visual attributes.

Parameters

signature1
The signature of the comparison image (the image with which other images are being compared to test for matches). Data type is RAW(2000).

signature2
The signature of the image being compared with the comparison image. Data type is RAW(2000).

weightstring
A list of weights to apply to each visual attribute. Data type is VARCHAR2.

The following attributes can be specified, with a value of 0.0 specifying no importance and a value of 1.0 specifying highest importance:

Attribute	Description
globalcolor	The weight value (0.0 to 1.0) assigned to the global color visual attribute. Data type is NUMBER. Default is 0.0.
localcolor	The weight value (0.0 to 1.0) assigned to the local color visual attribute. Data type is NUMBER. Default is 0.0.
texture	The weight value (0.0 to 1.0) assigned to the texture visual attribute. Data type is NUMBER. Default is 0.0.

Attribute	Description
structure	The weight value (0.0 to 1.0) assigned to the structure visual attribute. Data type is NUMBER. Default is 0.0.
facial	The two signatures are Viisage facial signatures. When comparing facial signatures, no other attributes can be included in the weightstring. Data type is NUMBER, and must be set to 1 if used.

Note: When specifying parameter values that include floating-point numbers, you should 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.

Returns

A FLOAT value between 0.0 and 100.0, where 0.0 is identical and 100.0 is totally different.

Usage

Before the Score( ) operator can be used, the image signatures must be created with the Analyze( ) operator.

The Score( ) operator can be useful when an application wants to make finer distinctions about matching than the simple Yes or No returned by the Similar( ) operator. For example, using the number for weighted sum returned by Score( ), the application might assign each image being compared to one of several categories, such as definite matches, probable matches, possible matches, and non-matches. The Score( ) operator can also be useful if the application needs to perform special processing based on the degree of similarity between images.

The weights supplied for the four visual attributes are normalized before processing such that they add up to 100 percent. To avoid confusion and meaningless results, you should develop a habit of always using the same scale, whether 0 to 100 or 0.0 to 1.0.

You must specify at least one of the four image attributes or the facial attribute, in the weightstring. You cannot combine the facial attribute with any of the other attributes.

Examples

Example 1: The following example finds the weighted sum of the distances between image 1 and the other images in the stockphotos table, using the following weights for the visual attributes:

- Global color: 0.2
- Local color: 0.2
- Texture: 0.1
- Structure: 0.5

Example 2: This example assumes that the signatures were created using the Analyze( ) operator and they are stored in the database.

```
DECLARE
   weighted_sum     NUMBER;
BEGIN
   SELECT Q.photo_id,
          ORDSYS.VIR.Score(S.photo.signature,
                                Q.photo.signature,
                                'globalcolor="0.2"
                                 localcolor="0.2"
                                 texture="0.1"
                                    structure="0.5"') weighted_sum
   FROM stockphotos Q, stockphotos S
     WHERE S.photo_id=1 and Q.photo_id !=S.photo_id;
END;
```

The following shows possible output from the previous example. The first image has the lowest score, and therefore is the best match of the test image (`photo_id=1`). Changing the weights used in the scoring would lead to different results.

```
PHOTO_ID   WEIGHTED_SUM
---------- -------------
         2    3.79988
         3    76.0807
         4    47.8139
         5    80.451
         6    91.2473
5 rows selected.
```

Similar( ) Operator

Format

```
Similar(signature1 IN RAW
        signature2 IN RAW,
        weightstring IN VARCHAR2
        threshold   IN FLOAT);
```

where weightstring is: 'globalcolor="val" localcolor="val" texture="val" structure="val"'

or: 'facial=1'

Description

Determines whether or not two images match. Specifically, compares the signatures of two images, computes a weighted sum of the distance between the two images using weight values for the visual attributes, compares the weighted sum with the threshold value, and returns the integer value 1 if the weighted sum is less than or equal to the threshold value. Otherwise, the Similar( ) operator returns 0.

Parameters

signature1
The signature of the comparison image (the image with which other images are being compared to test for matches).

signature2
The signature of the image being compared with the comparison image.

weightstring
A list of weights to apply to each visual attribute. Data type is VARCHAR2.

The following attributes can be specified, with a value of 0.0 specifying no importance and a value of 1.0 specifying highest importance:

Attribute	Description
globalcolor	The weight value (0.0 to 1.0) assigned to the global color visual attribute. Data type is NUMBER. Default is 0.0.

Attribute	Description
localcolor	The weight value (0.0 to 1.0) assigned to the local color visual attribute. Data type is NUMBER. Default is 0.0.
texture	The weight value (0.0 to 1.0) assigned to the texture visual attribute. Data type is NUMBER. Default is 0.0.
structure	The weight value (0.0 to 1.0) assigned to the structure visual attribute. Data type is NUMBER. Default is 0.0.
facial	The two signatures are Viisage facial signatures. When comparing facial signatures, no other attributes can be included in the weightstring. Data type is NUMBER, and must be set to 1 if used.

Note: When specifying parameter values that include floating-point numbers, you should 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.

threshold

The value with which the weighted sum of the distances is to be compared. If the weighted sum is less than or equal to the threshold value, the images are considered to match. The range of this parameter is from 0.0 to 100.0.

Returns

An integer value of 0 (not similar) or 1 (match).

Usage

Before the Similar( ) operator can be used, the image signatures must be created with the Analyze( ) operator.

The Similar( ) operator is useful when the application needs a simple Yes or No for whether or not two images match. The Score( ) operator is useful when an application wants to make finer distinctions about matching or to perform special processing based on the degree of similarity between images.

The weights supplied for the four visual attributes are normalized before processing such that they add up to 100 percent. To avoid confusion and meaningless results, you should develop a habit of always using the same scale, whether 0 to 100 or 0.0 to 1.0.

You must specify at least one of the four image attributes, or the facial attribute, in the weightstring. You cannot combine the facial attribute with any of the other attributes.

Examples

This example checks the first image against all the other images in the table and determines if there are any matches, using a threshold value of 75 and the following weights for the visual attributes:

- Global color: 20
- Local color: 20
- Texture: 10
- Structure: 50

```
BEGIN
   SELECT Q.photo_id FROM stockphotos Q, stockphotos S
      WHERE S.photo_id=1 and Q.photo_id != S.photo_id
      AND  ORDSYS.VIR.Similar(S.photo.signature,
                                    Q.photo.signature,
                                   'globalcolor="20"
                                    localcolor="20"
                                    texture="10"
                                        structure="50"', 75)=1;
END;
```

The following shows a possible output from this example. See the Examples section of the Score( ) operator for a different way of viewing the results using the same images and weights.

```
PHOTO_ID
----------
         2
         4
2 rows selected.
```

F

Error Messages

VIR-01001, "ANALYZE failed to generate the SIGNATURE"

Cause: The Analyze function could not generate the signature.

Action: Verify that the image being analyzed is valid.

VIR-01002, "SCORE failed to compare the SIGNATURES"

Cause: The Score function could not compare the two signatures.

Action: Verify that the signatures were generated correctly.

VIR-01003, "SIGNATURE buffer too small"

Cause: The signature being generated is larger than the storage allocated to receive it.

Action: Allocate more space for the signature, reduce the complexity of the image being analyzed, or crop the image to remove extraneous features.

VIR-01005, "empty or null attribute string"

Cause: An empty or null weightstring was passed to the Score or Similar function.

Action: Refer to the Oracle Visual Information Retrieval documentation for a description of the correct usage and syntax for the weightstring attribute.

VIR-01006, "invalid attribute value"

Cause: An invalid value was found while parsing the weightstring attribute for the Score or Similar functions.

Action: Correct the statement by using a valid attribute value. Refer to the Oracle Visual Information Retrieval documentation for a description of the correct usage and syntax for the Score and Similar weightstring attribute.

VIR-01007, "internal error"

Cause: An internal error has occurred.

Action: Contact Oracle Support Services.

VIR-01008, application specific message

Cause: A syntax error was found while parsing the weightstring attribute for the Score or Similar functions.

Action: Correct the statement by using valid parameter values. Refer to the Oracle Visual Information Retrieval documentation for a description of the correct usage and syntax of the weightstring attribute.

VIR-01009, "unable to read image data"

Cause: There is no image data in the image object.

Action: Refer to the Oracle Visual Information Retrieval documentation for information on how to populate image data into the image object.

VIR-01010, "SIGNATURE data has been corrupted or is invalid"

Cause: The data in the signature is not a valid Virage signature.

Action: Re-create the signature using the Analyze method.

VIR-01011, "SIGNATURE is in incorrect byte order"

Cause: The data in the signature may be a valid Virage signature, but it is in the incorrect byte order.

Action: Use the Convert method to change the byte order.

VIR-01012, "SIGNATURE conversion failed"

Cause: The data in the signature may not be a valid Viisage signature.

Action: Rewrite the signature with a valid Viisage signature.

VIR-01013, "invalid conversion operation"

Cause: The specified conversion operation is not valid.

Action: Refer to the Oracle Visual Information Retrieval documentation for a description of the correct usage and syntax for the Convert operation string.

VIR-01014, "specified weights are not valid"

Cause: The weights specified are not valid for Score. Both standard and facial weights were specified.

Action: Do not specify both standard and facial attribute weights.

VIR-01015, "no weights specified"

Cause: All weights passed were zero. At least one attribute must be weighted.

Action: Specify a weight for at least one attribute.

VIR-01016, "internal error during initialization"

Cause: An internal error has occurred while trying to initialize the Visual Information Retrieval image engine.

Action: Contact Oracle Support Services.

VIR-01017, "out of memory while analyzing image"

Cause: The external process agent has exhausted operating system memory while analyzing the image.

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

VIR-01018, "unable to convert SIGNATURE to native byte order"

Cause: The signature data might have been corrupted.

Action: Re-create the signature using the Analyze( ) method.

VIR-01019, "signature is not a Viisage SIGNATURE"

Cause: The incoming signature is not a Viisage signature.

Action: Rewrite the signature with a valid Viisage signature.

IMG-00001, "unable to initialize Oracle8*i* interMedia environment"

Cause: The image processing external procedure initialization process failed.

Action: Contact Oracle Support Services.

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 Oracle *inter*Media 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 4 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 4 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 Oracle *inter*Media 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 Support Services.

IMG-00512, "multiple incompatible scaling parameters found"

Cause: Multiple incompatible scaling parameters were found in the image process command string. With the exception of xScale and yScale, which can be used together in a process command string, scaling functions are mutually exclusive and cannot be combined.

Action: Remove scaling functions until only one remains (or two if they are xScale and yScale.)

IMG-00513, "missing value in scaling operation"

Cause: An incorrect number of values was used to specify image dimensions. fixedScale and maxScale require exactly two integer values for the x and y dimensions of the desired image.

Action: Use two values for fixedScale and maxScale.

IMG-00514, "extra value in scaling operation"

Cause: An incorrect number of values was used to specify image dimensions. FixedScale and maxScale require exactly two integer values for the x and y dimensions of the desired image.

Action: Use two values for fixedScale and maxScale.

IMG-00515, "incorrect number of input channels"

Cause: An incorrect number of values was used to specify input channels. inputChannels requires either one or three channel numbers for the grayscale or red, green, and blue channel assignments.

Action: Use either one or three values to specify the input channels.

IMG-00516, "default channel out of range"

Cause: An incorrect value was used to specify the default channel selection.

Action: Use a channel number that is less than or equal to the number of bands and greater than zero.

IMG-00517, "height or width not present in parameter string"

Cause: Height and/or width were not specified in the setProperties parameter string.

Action: Specify both the height and width.

IMG-00518, "invalid value for height or width"

Cause: Height and width must be positive integers.

Action: Specify both the height and width as positive integers.

IMG-00519, "illegal combination of parameters"

Cause: Other than height, width, dataOffset, and userString, no other parameters may be specified in the setProperties parameter string when CCITTG3 or CCITTG4 is used as the compressionFormat.

Action: Supply only the height and width when compressionFormat is either CCITTG3 or CCITTG4. The dataOffset and userString may optionally be supplied as well.

IMG-00520, "invalid value for numberOfBands"

Cause: NumberOfBands must be a positive integer.

Action: Specify numberOfBands as a positive integer.

IMG-00521, "invalid value for dataOffset"

Cause: dataOffset must be a positive integer.

Action: Specify dataOffset as a positive integer.

IMG-00530, "internal error while parsing command"

Cause: An internal error occurred while parsing the command passed to the image processing function or to the foreign image setProperties( ) function.

Action: Check that the command passed to the function. Refer to the Oracle *inter*Media documentation for a description of the correct usage and syntax for the image processing command string or the foreign image setProperties( )

function. If you are certain that your command is correct, then contact Oracle Support Services.

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 Oracle *inter*Media 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 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 source attribute is invalid.

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

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

Cause: The destination image source attribute is invalid.

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

IMG-00606, "unable to access image data"

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

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

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

Cause: The destination image source attribute is invalid.

Action: Ensure that the source attribute of the destination image is initialized correctly 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 image object.

Action: Refer to the Oracle *inter*Media documentation for information on how to populate image data into the image object.

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

Cause: The image processing external procedure initialization process failed.

Action: Contact Oracle Support Services.

IMG-00703, "unable to read image data"

Cause: There is no image data in the image object.

Action: Refer to the Oracle *inter*Media documentation for information on how to populate image data into the image object.

IMG-00704, "unable to read image data"

Cause: There is no image data in the image object.

Action: Refer to the Oracle *inter*Media documentation for information on how to populate image data into the image object.

IMG-00705, "unsupported or corrupted input format"

Cause: This is an internal error.

Action: Contact Oracle Support Services.

IMG-00706, "unsupported or corrupted output format"

Cause: This is an internal error.

Action: Contact Oracle 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 to write to destination image"

Cause: The destination image is invalid.

Action: Ensure that the source attribute of the destination image is initialized and that you have sufficient tablespace.

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

Cause: This is an internal error.

Action: Contact Oracle Support Services.

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

Cause: The destination image is invalid.

Action: Ensure that the source attribute of the destination image is initialized and that you have sufficient tablespace. Ensure that the row containing the destination image has been locked (this does not apply to temporary BLOBs).

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 *inter*Media documentation for supported formats.

IMG-00714, "internal error"

Cause: This is an internal error.

Action: Contact Oracle 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.

IMG-00716, "source image format does not support process options"

Cause: A request was made to apply a processing option not supported by the source image format.

Action: Refer to the Oracle *inter*Media documentation for a discussion of supported processing options.

IMG-00717, "destination image format does not support process options"

Cause: A request was made to apply a processing option not supported by the destination image format.

Action: Refer to the Oracle *inter*Media documentation for a discussion of supported processing options.

IMG-00718, "the same temporary LOB cannot be used as both source and destination"

Cause: A call was made to processCopy with the same temporary LOB being specified as both the source and destination.

Action: Specify a different LOB for parameter dest.

ORA-06502, "PL/SQL: numeric or value error"

Cause: The valid range for the threshold argument to the Similar( ) function is from 0.0 to 100.0.

Action: Correct the statement and try again.

Index

G

I

J

L

M