Excalibur Image DataBlade Module

User's Guide

Version 1.2
March 1999
Part No. 000-5356

Published by INFORMIX® Press

Informix Corporation
4100 Bohannon Drive
Menlo Park, CA 94025-1032

Documentation Team: Marjorie Larney, Clyanne Tuuri, Claire Mosher, Oakland Editing and Production

Contributors: Junlan Zheng, George McQuary

Table of Contents

Introduction

In This Introduction

This chapter introduces the *Excalibur Image DataBlade Module User's Guide*. Read this chapter for an overview of the information provided in this manual and for an understanding of the conventions used throughout.

About This Manual

This guide contains information about using the Excalibur Image DataBlade module with Informix Dynamic Server with Universal Data Option, hereafter called the Universal Data Option. The Excalibur Image DataBlade module adds custom data types and supporting routines to the server.

This section discusses the organization of the manual, the intended audience, and the associated software products that you must have to use the Excalibur Image DataBlade module.

Organization of This Manual

An overview chapter describes how the Excalibur Image DataBlade module works to support the storage and retrieval of images using the Universal Data Option.

This manual includes the following chapters:

- Chapter 1, "Overview of the Excalibur Image DataBlade Module," provides an overview of the product functionality and describes the Image Foundation component.
- Chapter 2, "Representing and Storing Images," describes how to represent images as smart large objects and how to store them in the database.

- Chapter 3, “Feature-Based Image Retrieval,” describes the combined feature extraction method and SQL syntax for image retrieval.
- Chapter 4, “Data Types,” documents the data types and structures defined in the Excalibur Image DataBlade module.
- Chapter 5, “SQL Reference,” documents the SQL functions provided by the Excalibur Image DataBlade module.
- Appendix A, “Supported Image Formats and Types,” documents the image types provided by the Excalibur Image DataBlade module.
- Appendix B, “Internal Data Structures, Data Types, and Functions,” describes data structures, data types, and functions to be used by developers who want to add new functions using Excalibur internal data structures.
- Appendix C, “C Code Examples,” contains the C code examples for the functions supplied for DataBlade module developers.

A glossary of terms specific to this DataBlade module follows the chapters, and a comprehensive index directs you to areas of particular interest.

Audience

This manual is written primarily for database users who want to store and retrieve images by searching on the content of the image using Universal Data Option. This manual shows developers how to build or modify feature extractor functions.

Additionally, the appendixes describe functions, data types, and structures that DataBlade module developers can use to build:

- image-based DataBlade modules using the data types and routines provided by the Excalibur Image DataBlade module.
- non-image-based DataBlade modules using the method and data types supplied by the Excalibur Image DataBlade module.

Software Dependencies

To use this manual, you must be using Universal Data Option. Check your release notes to determine which versions of Universal Data Option are compatible with this release of the Excalibur Image DataBlade module. In addition, you need to have the Excalibur Image DataBlade module and the Informix Large Object Locator DataBlade module installed and registered in all databases where you intend to store images for processing by the Excalibur Image DataBlade module.

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

The following conventions are covered:

- Typographical conventions
- Icon conventions

Typographical Conventions

This manual uses the following standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.

(1 of 2)

Convention	Meaning
italics	Within text, new terms and emphasized words appear in italics. Within syntax diagrams, values that you are to specify appear in italics.
boldface	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
`monospace`	Information that you enter on the computer and information that the product displays on your screen appear in a monospace typeface. Examples are always shown in monospace font.

(2 of 2)

Tip: *The text and many of the examples in this manual show function and data type names in mixed case (uppercase and lowercase letters). Because the Informix Dynamic Server database management system is case insensitive, you do not need to enter function names exactly as shown: you can use uppercase letters, lowercase letters, or any combination of the two.*

Icon Conventions

Comment icons identify warnings, important notes, or tips.

Icon	Description
	The *warning* icon identifies vital instructions, cautions, or critical information.
	The *important* icon identifies significant information about the feature or operation that is being described.
	The *tip* icon identifies additional details or shortcuts for the functionality that is being described.

This information is always displayed in italics.

Additional Documentation

This section describes the following parts of the documentation set:

- Printed documentation
- Release notes and documentation notes

Printed Documentation

The printed documentation for this product consists of one manual, the *Excalibur Image DataBlade Module User's Guide*. This manual introduces the Excalibur Image DataBlade module, including the data types and functions. It describes how the DataBlade module works to support the storage and retrieval of images. The manual also includes a glossary of Excalibur Image DataBlade module terms.

The following Informix manuals complement the information in this manual:

- *Informix Large Object Locator DataBlade Module User's Guide*
- *DataBlade Developers Kit User's Guide*
- *DataBlade Module Installation and Registration Guide*
- *DataBlade API Programmer's Manual*
- *Informix Guide to SQL: Reference*
- *Informix Guide to SQL: Syntax*
- *Informix Guide to SQL: Tutorial*

Release Notes and Documentation Notes

In addition to the Informix set of manuals, the following on-line files, located in the **$INFORMIXDIR/extend** directory, supplement the information in this manual.

On-Line File	Purpose
Release notes	Describe feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
Documentation notes	Describe features not covered in the manuals or modified since publication.

Please examine these files because they contain vital information about application and performance issues.

Informix Welcomes Your Comments

Let us know what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Include the following information:

- The name and version of the manual you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
Technical Publications
300 Lakeside Dr., Suite 2700
Oakland, CA 94612

If you prefer to send electronic mail, our address is:

doc@informix.com

We appreciate your suggestions.

Chapter

1

Overview of the Excalibur Image DataBlade Module

In This Chapter

This chapter provides an overview of the Excalibur Image DataBlade module and how it supports image storage and retrieval.

What Is the Excalibur Image DataBlade Module?

The Excalibur Image DataBlade module combines Excalibur image technology with the Universal Data Option to store, retrieve, and search images in a database. With the Excalibur Image DataBlade module and the Universal Data Option, you can use SQL statements to store a group of images in a database and retrieve them with a content-based search. Images are described by content attributes in the database, allowing you to select an image and search the database to locate similar images.

The image technology contained in the Excalibur Image DataBlade module is based on feature-vector extraction search techniques. The Excalibur Image DataBlade module allows you to perform content-based searches of groups of images by reference to the binary patterns in feature vectors.

Prerequisites

The Excalibur Image DataBlade module requires the following software:

- Informix Dynamic Server with Universal Data Option
- Informix Large Object Locator DataBlade module (included with the server)
- Images stored in supported raster image formats (see "Supported Image Formats" on page A-1 for more information)

Features and Functionality

With the Excalibur Image DataBlade module, you can:

- store and retrieve images in the database.

  When an image is stored in a database table, a *feature extractor function* can be used to record the characteristics of the image in an array of bits called a *feature vector*. The feature vector and the image are stored in separate columns, providing content-based access to the images in the database table.

- search for matching and similar images in the database.

  When images are retrieved, searching is made on the feature vectors extracted beforehand. The database table is searched for images with similar vectors, and a ranked list of matching images is returned.

Figure 1-1 on page 1-4 illustrates the functionality of the Excalibur Image DataBlade module.

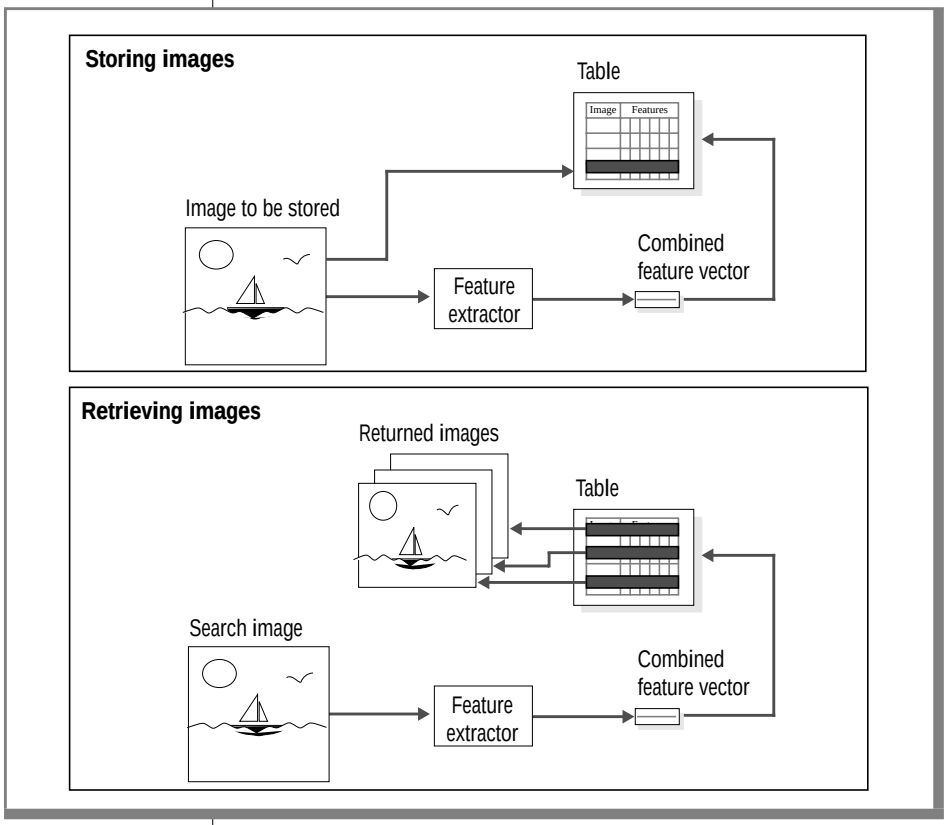


Figure 1-1
Storing and Retrieving Images

Excalibur Image DataBlade Module

The Excalibur Image DataBlade module provides storage, retrieval, and search capabilities for images. This includes image manipulation and I/O routines and feature extraction to store and retrieve images. The content of the Excalibur Image DataBlade module is shown in the following diagram.

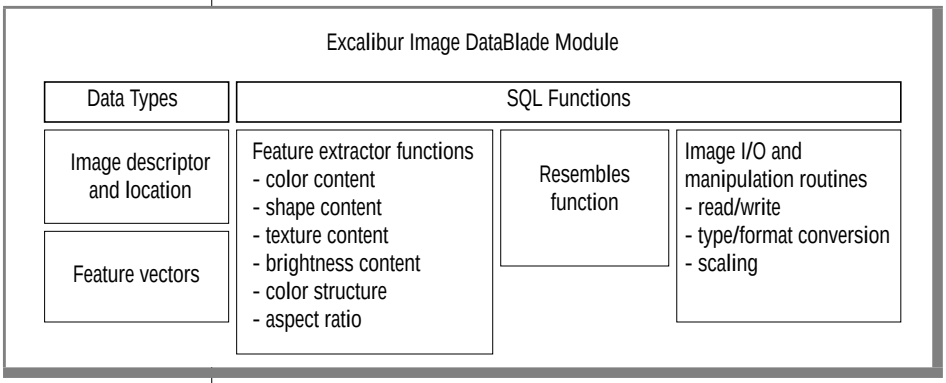


Figure 1-2
Excalibur Image DataBlade Module

The following sections describe the content of the Excalibur Image DataBlade module.

Data Types and I/O Routines

The Excalibur Image DataBlade module provides you with data types and routines for image I/O and image manipulation.

Using the features of the Excalibur Image DataBlade module, you can:

- store images in their native format (such as TIFF or GIF) while you process and retrieve them using the Excalibur Image DataBlade module.
- read and write images to and from database row types and data structures in the database server's shared memory. The image manipulation routines allow you to access and change the size, format, and type attributes of images stored in your database.

Image Search and Retrieval

The Excalibur Image DataBlade module lets you:

- extract feature vectors from images and store them in a table.
- retrieve images based on image features such as color, texture, and shape.

To retrieve and search images, the Excalibur Image DataBlade module uses the following functions:

- **Feature extractor function.** The **GetFeatureVector()** function extracts six features of an image in a combined vector for comparisons by the **Resembles()** function.

  To extract the combined feature vector in a table of stored images, you apply the **GetFeatureVector()** function on the stored images in the table.

  For more information about the feature extraction, see "Extracting Feature Vectors from Images" on page 3-5.

- **Resembles() function.** The **Resembles()** function performs a comparison on the combined feature vectors from two images. The **Resembles()** function applies the relative weight assigned to each of the six features in the combined feature vector to compute a resemblance score.

  For more information on the **Resembles()** function, see "Using the Resembles() Function" on page 3-8.

Using the Excalibur Image DataBlade Module

Storing and retrieving your images using the Excalibur Image DataBlade module involves the following basic tasks:

- Creating a database table for your images
  - Creating a column for holding images in a database table
  - Creating a column for holding feature vectors in a database table
- Inserting images into the database
- Extracting feature vectors

- Searching for images in the database

The following subsections provide a brief overview of how you can store and retrieve images using the Excalibur Image DataBlade module.

Creating the Database Table

The following example SQL statement creates a database table for your images that contains a column of data type IfdImgDesc:

```
CREATE TABLE gallery
      ( img_id  int,
        image   IfdImgDesc,
        fv      IfdFeatVect
      );
```

The Excalibur Image DataBlade module supplies this data type along with routines for manipulating your images. See Chapter 4, "Data Types," for more information.

Inserting Images into the Database

To insert images into a database table, use the following SQL statement:

```
INSERT INTO gallery (img_id, image) VALUES
      (1, IfdImgDescFromFile('/tmp/01.tif'));

  INSERT INTO gallery (img_id, image) VALUES
      (2, IfdImgDescFromFile('/tmp/02.bmp'));
```

Extracting Feature Vectors from Images

To extract and store feature vectors in the database table, you can use the following SQL statement:

```
UPDATE gallery SET fv = GetFeatureVector(image)
      WHERE fv IS NULL;
```

Searching for Images in the Database

To search the images in your database, locate those that match a selected *search image*. Specify the **Resembles()** function in the WHERE clause of the SQL SELECT statement:

```
SELECT img_id, rank FROM gallery
       WHERE Resembles(fv,

GetFeatureVector(IfdImgDescFromFile('/tmp/01.tif')),
                              0.0, 1, 1, 1, 0, 0, 0, rank #REAL)
       ORDER BY rank;
```

The *rank* expression must always be specified and is always returned.

Chapter

2

Representing and Storing Images

In This Chapter

This chapter provides information on how to represent and store images in a database and how to reference images in external files. The following topics are covered:

Representing Images as Smart Large Objects

You store your images in smart large objects to ensure recovery. You use the IfdImgDesc data type to define a column into which you load your images, one image per row. See "The IfdImgDesc Data Type" on page 4-4 for more information.

Using the FileToBlob() Function

You must have at least one sbspace in which to store images as smart large objects. You can store images in an sbspace that you name when you create your table or in the default space. Informix recommends the former.

Use the **onspaces** utility to create an sbspace. See the *Administrator's Guide* for your Informix database server for details.

For more information on the BLOB data type, refer to the *Informix Large Object Locator DataBlade Module User's Guide*.

Here is an example of how to insert an image into an **IfdImgDesc** column using the **FileToBlob()** function:

```
INSERT INTO gallery (img_id, image) VALUES
          (3,
          ROW(ROW(ROW('ifx_blob', FileToBlob('/tmp/03.jpg',
                                             'server'),
                        NULL::lvarchar)::lld_locator,
                    NULL::lvarchar)::IfdLocator,
               NULL::lvarchar, NULL::lvarchar,
               NULL::integer, NULL::integer,
               NULL::lvarchar)::IfdImgDesc
          );
```

Using the ReadImg() Function

Here is an example of how to insert an image into an **IfdImgDesc** column using the **ReadImg()** function:

```
INSERT INTO gallery (img_id, image) VALUES
          (1,
           ReadImg(ROW(ROW('ifx_file', NULL::lld_lob,
                           '/tmp/01.tif'),
                       NULL::lvarchar)::IfdLocator,
               ROW(LLD_Create(ROW('ifx_blob', NULL::lld_lob,
                              NULL::lvarchar)::lld_locator),
                       NULL::lvarchar)::IfdLocator,
                   NULL, NULL, 0)
          );
```

Referencing Images as External Files

If your images reside in an external file system, you can choose to build an image database to manage them. You do not need to move the image contents into the database but can use references to your images in external files.

If you store images in external files, you do not have to configure an sbspace. You can store the pathnames of your images in an IfdImgDesc row type as references to your external image files.

If a sample image used as search criterion resides in a database table rather than an external file, it is important to precompute its feature vector and use that in the query. Otherwise, the feature vector is recomputed for every row in the scanned tables.

Using a Nested Row Constructor

A nested row constructor can contain additional information about an image. Usually, all the other elements in the image descriptor row type are not needed and are often set to `NULL`, even on insertion.

The following example uses a nested row constructor:

```
INSERT INTO gallery (img_id, image) VALUES
          (3,
           ROW(ROW(ROW('ifx_file', NULL::LLD_Lob,
                       '/tmp/03.jpg')::lld_locator,
                   NULL::lvarchar)::IfdLocator,
               NULL::varchar, NULL::varchar,
               NULL::integer, NULL::integer,
               NULL::varchar)::IfdImgDesc
          );
```

Using the IfdImgDescFromFile() Function

The **IfdImgDescFromFile()** function creates an IfdImgDesc value from the image whose path is specified in the path parameter. If the specified image resides in an operating system file accessible from the database server, this function can be used in place of the nested row constructor commonly used to create IfdImgDesc values. The **IfdImgDescFromFile()** function takes only the path as an input argument.

The following example uses the **IfdImgDescFromFile()** function:

```
INSERT INTO gallery (img_id, image) VALUES
          (3, IfdImgDescFromFile('/tmp/03.jpg'));
```

Chapter

3

Feature-Based Image Retrieval

In This Chapter

This chapter shows how the Excalibur Image DataBlade module uses features to search for and retrieve images stored in a database according to their color content and structure, shape content, texture content, brightness structure, and aspect ratio.

What Is Feature-Based Retrieval?

The Excalibur Image DataBlade module provides for feature extraction, which allows you to perform content-based searches on collections of images. Feature extraction analyzes an image to determine its characteristics and reduces the image to a condensed descriptor, a *feature vector*.

The Excalibur Image DataBlade module provides *feature extractor functions* to encode the important features of your images. A feature extractor function records the presence or absence of a feature in a string of bits, the feature vector. The Excalibur Image DataBlade module also provides a data type (IfdFeatVect) for storing feature vectors.

For more information on the IfdFeatVect data type, see Chapter 4, “Data Types.”

Extracting Features

A *feature extractor* is a function that encodes important attributes of an image into a feature vector. Each bit in the feature vector indicates the presence or absence of a certain feature in the image.

The Excalibur Image DataBlade module uses a combination of feature vectors to retrieve images from a database by searching the database for matching images. Feature vectors provide the following advantages:

- They allow you to retrieve images based on their content and structure.
- They are condensed and require less storage than the actual image data.

Additionally, creating a feature vector allows you to normalize an image by removing those features that are not important to a particular application.

The Feature Vector

To use the feature extraction method, you need to create, populate, and update a column to hold the feature vectors that include the following six combined features:

- Color Content

  This feature is a measure of the colors in an image, including their relative proportions. The position of the colors does not matter.

- Shape Content

  This feature is a measure of the relative orientation, curvature, and contrast of lines in an image. The color, position, and absolute orientation of the lines does not matter. This feature is most effective when the lines are crisp and clear.

- Texture Content

  This feature is a measure, on a small scale, of the flow and roughness of an image. The color, position, and orientation of these features does not matter.

- Brightness Structure

  This feature is a measure of the brightness at each point in the image.

- Color Structure

  This feature is a measure of the hue, saturation, and brightness at each point in the image.

- Aspect Ratio

  This feature is a measure of the ratio of the width to the height in the image.

The feature vectors created by feature extractor functions are of the data type **IfdFeatVect,** an opaque data type. The Excalibur Image DataBlade module uses the feature vectors stored in the **IfdFeatVect** column. The functions are not called directly but are invoked in the SQL SELECT or INSERT statements.

The bits in a given feature vector represent the presence or absence of such features as colors or edges. If the feature extractor function finds an image that contains a given feature, it sets the bit that corresponds to the feature vector to `1`; if not, the feature extractor sets the bit to `0`.

Extracting Feature Vectors from Images

Before you can extract feature vectors from images, you need to create a table of images and include an **IfdFeatVect (fv)** column with the image table.

If you already created a table and inserted images using the examples in a previous chapter, then you should skip the next two examples. Go directly to the third example for filling in your feature vectors with `UPDATE`.

You can use the following example to create your table and its feature vectors:

```
CREATE TABLE gallery
(   img_id  integer PRIMARY KEY,
    image   IfdImgDesc,
    fv      IfdFeatVect
);
```

Then, to load your image data, insert your image first and extract the feature vector later. You can use the following example to load your data:

```
INSERT INTO gallery (img_id, image) VALUES (1,
ROW(ROW(ROW('ifx_file', NULL::LLD_Lob,
            '/tmp/01.tif')::lld_locator,
            NULL::lvarchar)::IfdLocator,
            NULL::varchar, NULL::varchar,
            NULL::integer, NULL::integer,
            NULL::varchar)::IfdImgDesc);
```

Using the GetFeatureVector() Function

This function extracts the features of an image for later comparison by the **Resembles()** function. The relative weight assigned to each of the features in computing an image's resemblance score is controlled by the function parameters of the **Resembles()** function. Scoring is done with weighted combinations of the six features.

RetinaWidth and *RetinaHeight* are used to scale the original images to smaller images, greatly reducing the amount of memory used and the time required for retrieving each image, without significant loss of resolution.

The following example uses the default *RetinaWidth* and *RetinaHeight:*

```
UPDATE gallery SET fv = GetFeatureVector(image)
          WHERE fv IS NULL;
```

The following example uses the *RetinaWidth* and *RetinaHeight* that you specify:

```
UPDATE gallery SET fv = GetFeatureVector(image, 128, 128)
          WHERE fv IS NULL;
```

After loading each image you intend to use, fill in the feature vector column. The following example uses a sample dimension of 128 for the *RetinaHeight* and *RetinaWidth* parameters and updates the feature vectors associated with the image of ID 1:

```
UPDATE gallery SET fv = GetFeatureVector(image, 128, 128)
          WHERE img_id = 1;
```

For later insertions, fill in the feature vector for other images that you want to add to the table. Use `NULL` to generate new feature vectors for images without feature vectors previously extracted, as in the example that follows:

```
UPDATE gallery SET fv = GetFeatureVector(image)
          WHERE fv IS NULL;
```

Keeping Feature Vectors Consistent with Images

If you change an image in your table, you also need to update the feature vectors associated with that image. You can update the feature vectors automatically with a trigger or you can use an update statement.

The example that follows uses a trigger to update feature vectors automatically:

```
CREATE TRIGGER insert_fv INSERT ON gallery
          REFERENCING NEW AS new
          FOR EACH ROW
              ( UPDATE gallery
                  SET fv = GetFeatureVector(new.image)
                WHERE img_id = new.img_id
              );
```

The example that follows uses an update statement to update the feature vectors associated with an image:

```
UPDATE gallery SET fv = GetFeatureVector(image, 128, 128)
          WHERE img_id = 1;
```

Checking the Feature Vector Version

The **Version()** function returns an integer that is related to the version of the Excalibur Image DataBlade module used to build the IfdFeatVect vectors. The primary purpose of this function is to aid integrity checking and problem diagnosis. You can use the **Version()** function to check incompatible or corrupted feature vectors.

The following example checks the version:

```
SELECT Version(fv) FROM gallery;
```

You can use the following example to detect all the image rows with feature vectors that are incompatible with Version 1 (the current version):

```
SELECT * FROM gallery WHERE Version(fv) <> 1;
```

Searching for Images

The Excalibur Image DataBlade module uses the **Resembles()** function to perform a comparison on the combined feature vectors of two images. The **Resembles()** function applies the relative weight assigned to each of the six features in the combined feature vector to compute a resemblance score.

Using the Resembles() Function

The **Resembles()** function returns a Boolean value indicating whether the resemblance score of two different IfdFeatVect values exceeds a user-defined threshold. **Resembles()** also provides the actual score in an "out" parameter. The score given to the different features determines each feature's relative contribution to the final score, which is a real number between `0.0` and `1.0`. Setting the threshold above `0.0` reduces the result set and saves time by truncating comparisons that are found to be unable to reach this threshold earlier in the process. Truncated comparisons result in a score of `0`.

The weight factors are relative to each other and only determine a feature's relative contribution to the final score. For two given IfdFeatVect values, the following array of weight factors, in sets of six, all return exactly the same resemblance score:

```
1,1,2,1,1,1
2,2,4,2,2,2
4,4,8,4,4,4
10,10,20,10,10,10
```

The sum of all weight factors must not exceed 100. There is no need for them to add up to 100, because they are not percentages, but their sum must not exceed 100.

Any feature whose weight is set to `0` does not contribute to the final resemblance score.

Searching with an Image in the Database

The fastest method for searching is to use an image already in the database, with its feature vectors available. You can use the following examples for searches that are efficient.

Matching the Feature Vectors Directly

The following example matches the feature vectors directly:

```
SELECT g.img_id, rank FROM gallery g, gallery s
          WHERE Resembles(g.fv, s.fv,
                          0.0, 1, 1, 1, 0, 0, 0, rank #REAL)
                AND s.img_id = 3
          ORDER BY rank;
```

Using Six Features and 80% Resemblance

The following example uses all six feature attributes and a threshold of 80%, ranked by closeness:

```
SELECT img_id, rank FROM gallery
WHERE Resembles(fv,
    GetFeatureVector(IfdImgDescFromFile('/tmp/03.jpg')),
0.80, 10, 30, 40, 10,
    5, 5, rank #REAL)
ORDER BY rank;
```

Using Six Features Weighted Equally

The following example uses all six features weighted equally and a threshold of 80%; the sample image is image 3 in the table:

```
SELECT g.img_id, rank FROM gallery g, gallery s
WHERE Resembles(g.fv, s.fv,
0.80, 1,1,1,1,1,1,rank #REAL)
AND s.img_id = 3
```

Using 95% Resemblance to Texture

The following example uses a threshold of 95% for texture:

```
SELECT Img_id, rank FROM gallery
WHERE Resembles(fv,

GetFeatureVector(IfdImgDescFromFile('/tmp/03.jpg'), 80, 80),
0.95,
            0, 0, 1, 0, 0, 0, rank #REAL)
ORDER BY rank;
```

Searching with an Image in an External File

You can search efficiently with an image in an external file. However, it is important that you create an IfdImgDesc value and avoid using a nested row constructor.

Using IfdImgDescFromFile()

The following example shows the creation of an IfdImgDesc value using **IfdImgDescFromFile().**

```
SELECT img_id, rank FROM gallery
          WHERE Resembles(fv,

GetFeatureVector(IfdImgDescFromFile('/tmp/03.jpg')),
                              0.0, 1, 1, 1, 0, 0, 0, rank #REAL)
          ORDER BY rank;
```

Using a Nested Row Constructor

The following example shows the creation of an IfdImgDesc value using a nested row constructor:

```
SELECT img_id, rank FROM gallery
          WHERE Resembles(fv,
              GetFeatureVector(ROW(ROW(ROW('ifx_file',
NULL::lld_lob,
                                                  '/tmp/03.jpg'),
                                             NULL::lvarchar),
                                  NULL::lvarchar, NULL::lvarchar,
                                     NULL::integer, NULL::integer,
                                     NULL::lvarchar)::IfdImgDesc),
                                  0.0, 1, 1, 1, 0, 0, 0, rank #REAL)
          ORDER BY rank;
```

Tip: *You should avoid using the nested row method, because the* ***GetFeatureVector*** *expression is evaluated repeatedly, once for every row in the table gallery, even though the function is declared to be NOT VARIANT. This is because its argument,* `row(row(row(...)))`*, is itself considered VARIANT by the server.*

Two Examples Illustrating Retrieval

The two examples that follow use images supplied with the Excalibur Image DataBlade module. The first example illustrates the recommended procedure, because it is faster than the procedure in the second example.

To use a reference to an external file

1. Create a table gallery:

```
CREATE TABLE gallery (img_id int, image ifdimgdesc, fv
IfdFeatVect);
```

2. Insert an image stored in an ifx_file lob loc:

```
INSERT INTO gallery (img_id, image) VALUES (1,
    IfdImgDescFromFile('/tmp/01.tif'));

INSERT INTO gallery (img_id, image) VALUES (3,
    IfdImgDescFromFile('/tmp/03.jpg'));
```

3. Update to extract feature vectors:

```
UPDATE gallery
    set fv = GetFeatureVector(image)
    where fv IS NULL;
```

4. Find the matching image. List all images by their score:

```
SELECT img_id, rank FROM gallery
WHERE Resembles(fv,
    GetFeatureVector(IfdImgDescFromFile('/tmp/03.jpg')),
    0.00, 1, 1, 1, 1, 1, 1, rank #REAL)
ORDER BY rank;
```

5. Find the matching image by eliminating those below the threshold:

```
SELECT img_id, rank FROM gallery
WHERE Resembles(fv,
    GetFeatureVector(IfdImgDescFromFile('/tmp/03.jpg')),
    0.80, 1, 1, 1, 1, 1, 1, rank #REAL)
ORDER BY rank;
```

To use images within a database

1. Make sure the administrator sets up an sbspace beforehand. Then, load images into gallery:

```
DROP TABLE gallery;

 CREATE TABLE gallery
  (img_id serial,
   image ifdimgdesc,
   fv IfdFeatVect);

  INSERT INTO gallery (img_id, image)  VALUES (1,
   ROW(ROW(ROW('ifx_blob',FileToBlob('/tmp/01.tif',
'server'),
   NULL::lvarchar)::lld_locator,NULL::lvarchar)::ifdlocator,
   NULL::lvarchar,
   NULL::lvarchar, NULL::integer, NULL::integer,
   NULL::lvarchar)::ifdimgdesc);

  INSERT INTO gallery (img_id, image)  VALUES (2,
   ROW(ROW(ROW('ifx_blob',FileToBlob('/tmp/02.bmp',
'server'),
   NULL::lvarchar)::lld_locator,NULL::lvarchar)::ifdlocator,
   NULL::lvarchar,
   NULL::lvarchar, NULL::integer, NULL::integer,
   NULL::lvarchar)::ifdimgdesc);

  INSERT INTO gallery (img_id, image)  VALUES (3,
   ROW(ROW(ROW('ifx_blob',FileToBlob('/tmp/03.jpg',
'server'),
   NULL::lvarchar)::lld_locator,NULL::lvarchar)::ifdlocator,
   NULL::lvarchar,
   NULL::lvarchar, NULL::integer, NULL::integer,
   NULL::lvarchar)::ifdimgdesc);

  INSERT INTO gallery (img_id, image)  VALUES (4,
   ROW(ROW(ROW('ifx_blob',FileToBlob('/tmp/04.gif',
'server'),
   NULL::lvarchar)::lld_locator,NULL::lvarchar)::ifdlocator,
   NULL::lvarchar,
   NULL::lvarchar, NULL::integer, NULL::integer,
   NULL::lvarchar)::ifdimgdesc);
```

2. Extract the feature vectors from the images:

```
SELECT img_id, rank FROM gallery
WHERE Resembles(fv,
    GetFeatureVector(IfdImgDescFromFile('/tmp/01.tif')),
    0.00, 1, 1, 1, 1, 1, 1, rank #REAL)
ORDER BY rank;
```

3. Find the matching image. List all images by their score:

```
SELECT img_id, rank FROM gallery
WHERE Resembles(fv,
    GetFeatureVector(IfdImgDescFromFile('/tmp/01.tif')),
    0.00, 1, 1, 1, 1, 1, 1, rank #REAL)
ORDER BY rank;
```

4. Find the image by eliminating those matches below the threshold:

```
SELECT img_id, rank FROM gallery
WHERE Resembles(fv,
    GetFeatureVector(IfdImgDescFromFile('/tmp/01.tif')),
    0.80, 1, 1, 1, 1, 1, 1, rank #REAL)
ORDER BY rank;
```

Chapter

4

Data Types

In This Chapter

This chapter documents the data types and structures defined in the Excalibur Image DataBlade module.

The Excalibur Image DataBlade module supplies the following data types:

- **IfdImgDesc**. A complex data type for storing images.
- **IfdLocator**. A complex data type for storing the location of images. The IfdLocator data type is a subordinate part of the IfdImgDesc data type and is dependent upon the Informix Large Object Locator DataBlade module.
- **IfdFeatVect**. A complex data type for containing the combined feature vector data.

The IfdImgDesc Data Type

The IfdImgDesc data type is a complex data type used to describe the attributes of an image. The IfdImgDesc data type defines a column into which you load your images, one image per row.

The IfdImgDesc data type allows you to maintain your images in their native file format and store them in locations supported by the Informix Large Object Locator DataBlade module, such as within a smart large object or in operating system files.

The following is the type definition for IfdImgDesc:

```
CREATE ROW TYPE IfdImgDesc
(
    location     IfdLocator,
    imgformat    varchar(25),
    imgtype      varchar(25),
    pixelheight  integer,
    pixelwidth   integer,
    params       varchar(128)
);
```

The following list describes the fields of the IfdImgDesc data type:

location Specifies the location of your image. This can be any location supported by the Informix Large Object Locator DataBlade module. The *location* field is an IfdLocator data type. This data type is documented in "The IfdLocator Data Type" on page 4-6.

imgformat Specifies the format of an image. For a description of the supported formats and file extensions, see "Supported Image Formats" on page A-1.

imgtype Specifies the image pixel type. For a description of supported image types, see "Supported Image Types" on page A-2.

pixelheight	Specifies the height of an image, in pixels.
pixelwidth	Specifies the width of an image, in pixels.
params	Supplied for other image-based DataBlade modules. No fixed format or protocol is required. The *params* field is not used by the Excalibur Image DataBlade module.

The IfdLocator Data Type

The IfdLocator data type is subordinate to the IfdImgDesc data type. It is used to provide uniform access to large objects, regardless of storage location.

The following is the type definition for IfdLocator:

```
CREATE ROW TYPE IfdLocator
(
    locator    lld_locator,
    lo_user    lvarchar(1024)
);
```

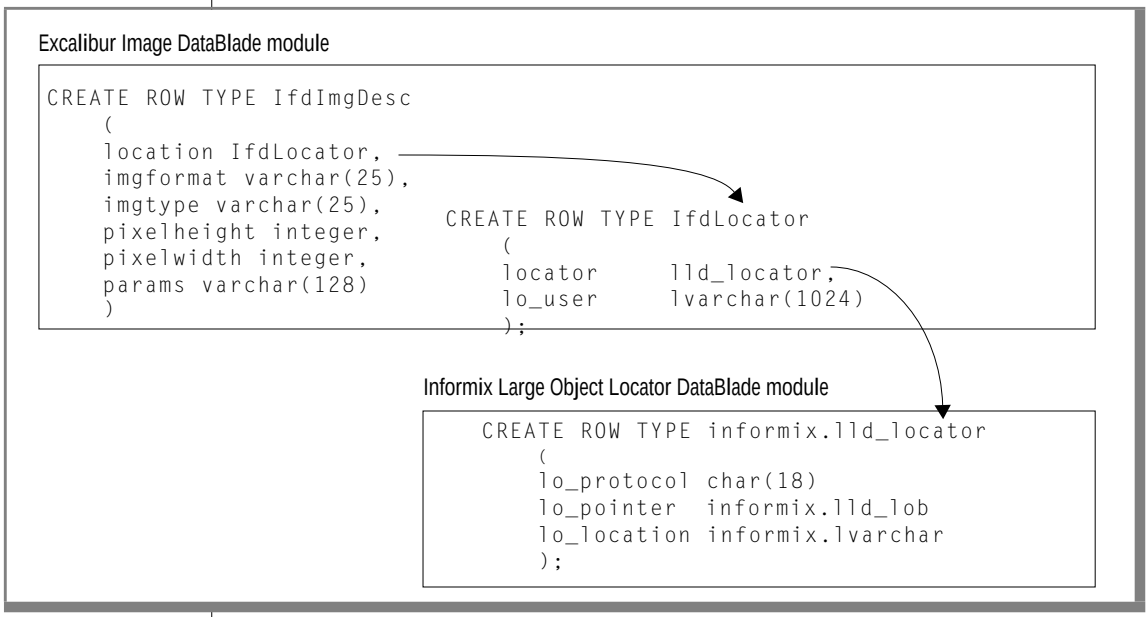
The following list describes the fields of the IfdLocator data type:

locator Specifies the database row data type defined by the Informix Large Object Locator DataBlade module to locate a large object.

lo_user Is available for users. The Excalibur Image DataBlade module does not use this parameter.

The relationship of the IfdLocator data type to the IfdImgDesc data type and the lld_locator data type is shown in the following figure.

Figure 4-1
Relationship of IfdImgDesc and LOB Locator Data Types



See the *Informix Large Object Locator DataBlade Module User's Guide* for more information on the lld_locator data type.

The IfdFeatVect Data Type

The IfdFeatVect data type is a complex data type that contains all the data for the combined feature vectors you use to store and retrieve your images. You use the **GetFeatureVector()** function to extract features from an image and to get return values to populate a column of type IfdFeatVect.

This data type is opaque and of variable length to accommodate future enhancements. In this release, the space actually used for every value is fixed at 376 bytes in addition to what is required for variable-length data.

The type definition for IfdFeatVect data type is as follows:

```
CREATE Opaque Type IfdFeatVect
(
    ALIGNMENT = 4,
    INTERNALLENGTH = VARIABLE,
    MAXLEN = 2048
);
```

Chapter

5

SQL Reference

In This Chapter

This chapter provides reference information on the SQL functions for reading, writing, and manipulating images.

All these functions are created and registered with the SQL CREATE FUNCTION statement in a registration file shipped with the Excalibur Image DataBlade module. More examples can be found in the examples file shipped with the Excalibur Image DataBlade module.

The following table lists the functions included in this chapter.

(1 of 2)

Resembles()	Scores resemblance of one feature vector to another based on an array of weights	page 5-19
ScaleImgBy()	Scales an image by a specified factor	page 5-21
ScaleImgTo()	Scales an image to a specified size	page 5-23
Version**()**	Returns an integer related to the version	page 5-25
WriteImg()	Writes an image from an IfdImgDesc row type to an IfdLocator destination	page 5-26

(2 of 2)

Tip: *The Excalibur Image DataBlade module also supplies functions intended primarily for DataBlade module developers. These functions, which read and write images to a C structure in shared memory and free the pointer to the structure, are documented in Appendix C, "C Code Examples."*

ConvertImgFormat()

Converts an image from one format to another.

Syntax

```
IfdImgDesc ConvertImgFormat(image, newformat)
    IfdImgDesc image;
    VARCHAR(25) newformat;
```

image Specifies the image to be converted.

newformat Specifies the result image format. Supported formats are EXIMG or one of the formats listed in "Supported Image Formats" on page A-1.

Usage

The **ConvertImgFormat()** function converts the image stored in the database to the specified format and updates the **imgformat** field of the IfdImgDesc data type to reflect the change.

Returns

On success, returns an IfdImgDesc row type that contains the image in the converted format. Images are returned as smart large objects regardless of the source image data type.

Example

```
UPDATE gallery
          SET image = ConvertImgFormat(image, 'TIFF')
          WHERE img_id = 3;
```

ConvertImgType()

Converts from one image type to another.

Syntax

```
IfdImgDesc ConvertImgType(image, newtype)
    IfdImgDesc image;
    VARCHAR(25) newtype;
```

image	Specifies the image whose image type is to be converted.
newtype	Specifies the result image type. For a list of supported image types, see "Supported Image Types" on page A-2.

Usage

The **ConvertImgType()** function converts an image to the specified image type and updates the **imgtype** field of the IfdImgDesc value to reflect the change.

Returns

On success, returns an IfdImgDesc object that contains the converted image type. Images are returned as smart large objects regardless of the source image data type.

Example

```
UPDATE gallery
          SET image = ConvertImgType(image, 'CTAB')
          WHERE img_id = 1;
```

GetFeatureVector()

Extracts the feature vector from an image.

Syntax—Default RetinaWidth and RetinaHeight (96x96)

```
GetFeatureVector( image IfdmgDesc )
```

Syntax—User-Specified RetinaWidth and RetinaHeight

```
GetFeatureVEctor( image IfdmgDesc,
    RetinaWidth integer,
    RetinaHeigth integer )
```

image Specifies the image to be processed.

integer Specifies a number in pixels.

Usage

The **GetFeatureVector()** function extracts features of an image for later use with the **Resembles()** function.

Returns

On success, returns values for feature vectors related to the retina width and height of the image.

IfdImgDescFromFile()

Creates an IfdImgDesc value from the image in a specified path.

Syntax

```
IfdImgDescFromFile(Path)
```

path Specifies the path of the image whose format is to be returned.

Usage

The **IfdImgDescFromFile()** function creates IfdImgDesc values from external files in preference to the more commonly used nested row constructor.

Returns

On success, returns a value from the image specified.

ImgCompression()

Returns the compression scheme used in the input image.

Syntax

```
VARCHAR(25) ImgCompression(image)
    IfdImgDesc image;
```

image Specifies the image whose compression scheme is to be returned.

Usage

The **ImgCompression()** function retrieves the compression scheme of the specified image. The following table summarizes the compression schemes returned by the **ImgCompression()** function.

Compression Scheme	Image Type	Definition
NOCOMP	BMP, GRAY, CTAB, RGB, HSV	Uncompressed image pixel data
PACKBITS	BMP, GRAY, CTAB, RGB, HSV	Pack bits
G31_TIFF	BMP	TIFF group 3, one-dimensional compression, no EOL, new rows begin on byte boundary
G31_T4	BMP	CCITT group 3, one-dimensional compression, with no padding of EOL to byte boundary
G31_T4_PAD	BMP	CCITT group 3, one-dimensional compression, with padding of EOL to byte boundary

(1 of 2)

Compression Scheme	Image Type	Definition
G32_T4	BMP	CCITT group 3, two-dimensional compression, with no padding of EOL to byte boundary
G42	BMP	CCITT group 4, two-dimensional compression
LZW_TIFF	BMP, GRAY, CTAB, RGB, HSV	TIFF LZW
LZW_PRED_TIFF	BMP, GRAY, CTAB, RGB, HSV	TIFF LZW with predictor

(2 of 2)

Returns

On success, returns a VARCHAR value that indicates the compression scheme of the original image.

Examples

```
SELECT ImgCompression(image) FROM gallery
          WHERE img_id = 1;
```

ImgFormat()

Returns the format of the specified image.

Syntax

```
VARCHAR(25) ImgFormat(image)
    IfdImgDesc image;
```

image Specifies the image whose format is to be returned.

Usage

The **ImgFormat()** function returns the format of the stored image.

For a list of supported image formats, see "Supported Image Formats" on page A-1.

Returns

On success, returns a character string that indicates the format of the specified image. If an invalid input IfdImgDesc image descriptor is received, returns null attribute values.

Example

```
SELECT ImgFormat(image) FROM gallery;
```

ImgHeight()

Returns the height (in pixels) of the specified image.

Syntax

```
INTEGER ImgHeight(image)
    IfdImgDesc image;
```

image Specifies the image whose height is to be returned.

Usage

The **ImgHeight()** function determines the height of a stored image.

Returns

On success, returns an integer indicating the height (in pixels) of the specified image. If an invalid input IfdImgDesc image descriptor is received, returns null attribute values.

Example

```
SELECT ImgHeight(image) FROM gallery;
```

ImgType()

Returns the image type of the specified image.

Syntax

```
VARCHAR(25) ImgType(image)
    IfdImgDesc image;
```

image Specifies the image whose image type is to be returned.

Usage

The **ImgType()** function retrieves the image type of a stored image.

Returns

On success, returns a character string indicating the image type of the specified image.

The returned value is one of the image types listed in "Supported Image Types" on page A-2.

If an invalid input IfdImgDesc image descriptor is received, returns null attribute values.

Example

```
SELECT ImgType(image) FROM gallery;
```

ImgWidth()

Returns the width (in pixels) of the specified image.

Syntax

```
INTEGER ImgWidth(image)
    IfdImgDesc image;
```

image Specifies the image whose width is to be returned.

Usage

The **ImgWidth()** function determines the width of a stored image.

Returns

On success, returns an integer indicating the width (in pixels) of the specified image. If an invalid input IfdImgDesc image descriptor is received, returns null attribute values.

Example

```
SELECT ImgWidth(image) FROM gallery;
```

ReadImg()

Reads an image and stores it in the IfdLocator destination parameter.

Syntax

```
IfdImgDesc ReadImg(source_image, dest_image, imgformat, imgtype,
frame_no)
    IfdLocator source_image;
    IfdLocator dest_image;
    VARCHAR(25) imgformat;
    VARCHAR(25) imgtype;
    INTEGER frame_no;
```

source_image	Specifies the source image.
dest_image	Specifies the destination location for the image.
imgformat	Specifies the image file format. Value can be AUTO, NULL, or one of the formats listed in "Supported Image Formats" on page A-1.
imgtype	Specifies the image type. Value can be AUTO, NULL, or one of the image types listed in "Supported Image Types" on page A-2.
frame_no	Specifies the file frame number (0 for single-frame images).

Usage

The **ReadImg()** function performs the following tasks:

1. Reads an image referenced in the source IfdLocator row.
2. Processes an image when the destination image format and image type are different from those of the source image.
3. Creates an IfdImgDesc row type.
4. Writes an image to the destination IfdLocator.
5. Returns the IfdImgDesc row type.

The Excalibur Image DataBlade module stores the image in the destination specified by the *locator* parameter of the IfdLocator data type. To store the image by reference to a file, set the *lo_protocol* field of the lld_locator structure to IFX_FILE, the *lo_pointer* to NULL::LLD_Lob, and the *lo_location* field to the pathname of the file. For more information, see "The IfdLocator Data Type" on page 4-6. For more information on setting values for the lld_locator data type, see the *Informix Large Object Locator DataBlade Module User's Guide*.

To determine the image format or image type of the image automatically, set the *imgformat* or *imgtype* parameter to AUTO. If you specify a different *imgformat* or *imgtype* parameter, the **ReadImg()** function converts the image format or image type and writes to the destination IfdLocator structure in the new image format or image type. The destination lld_locator structure contained in the IfdLocator structure must already exist. You can create the lld_locator structure using the **LLD_Create()** function.

If you specify NULL for both the *imgformat* and *imgtype* parameters, the image is copied from the source to the destination IfdLocator structure without being read into memory. This is useful for performing batch image inserts. The *pixelheight* and *pixelwidth* fields of the IfdImgDesc row type (in addition to *imgformat* and *imgtype*) are populated with null values. These fields can be updated later with the **ImgType()**, **ImgFormat()**, **ImgWidth()**, and **ImgHeight()** functions.

To assign a *dest_image* value in an Informix large object data type to a destination lld_locator structure, the *dest_image* parameter of the IfdLocator structure must be defined before it is passed to the **ReadImg()** function.

The *frame_no* parameter specifies the frame you want the **ReadImg()** function to read. The *frame_no* parameter works only for file formats that support multiple frames. The default value of *frame_no* is 0, which must be used for image formats that support a single frame.

Returns

On success, returns an image of data type IfdImgDesc.

Example

This example uses **ReadImg()** with five arguments:

```
INSERT INTO gallery (img_id, image) VALUES
          (1,
           ReadImg(ROW(ROW('ifx_file', NULL::lld_lob,
                           '/tmp/01.tif')::lld_locator,
                       NULL::lvarchar)::IfdLocator,
                   ROW(ROW('ifx_file', NULL::lld_lob,
                           '/tmp/tmpimg.tif')::lld_locator,
                       NULL::lvarchar)::IfdLocator,
                   NULL, NULL, 0)
          );
```

This image use **ReadImg()** from one location to another as a smart large object:

```
INSERT INTO gallery (img_id, image) VALUES
          (1,
           ReadImg(ROW(ROW('ifx_file',NULL::lld_lob,
                           '/tmp/01.tif')::lld_locator,
                          NULL::lvarchar)::IfdLocator,
              ROW(LLD_Create(ROW('ifx_blob', NULL::lld_lob,
                             NULL::lvarchar)::lld_locator),
                       NULL::lvarchar)::IfdLocator,
                   NULL, NULL, 0)
          );
```

This example uses **ReadImg()** to insert an image from a file to the database as an external file object with conversion:

```
INSERT INTO gallery (img_id, image) VALUES
          (1,
           ReadImg(ROW(ROW('ifx_file', NULL::lld_lob,
                           '/tmp/01.tif')::lld_locator,
                       NULL::lvarchar)::IfdLocator,
                   ROW(ROW('ifx_file', NULL::lld_lob,
                           '/tmp/01.tif')::lld_locator,
                       NULL::lvarchar)::IfdLocator,
                  'BMP', 'RGB', 0)
          );
```

This example uses **ReadImg()** to insert an image from a file to the database as a smart large object with conversion:

```
INSERT INTO gallery (img_id, image) VALUES
          (1,
           ReadImg(ROW(ROW('ifx_file',NULL::lld_lob,
                           '/tmp/01.tif')::lld_locator,
                       NULL::lvarchar)::IfdLocator,
              ROW(LLD_Create(ROW('ifx_blob', NULL::lld_lob,
                             NULL::lvarchar)::lld_locator),
                       NULL::lvarchar)::IfdLocator,
                    'BMP', 'RGB', 0)
          );
```

Resembles()

Scores resemblance of one feature vector to another based on an array of weights.

Syntax

```
Resembles(Fv1, Fv2, Threshold, ColorContentWt, ShapeContentWt,
TextureContentWt, BrightnessStructureWt, ColorStructureWt,
AspectRatioWt, Score)
    IfdFeatVect          Fv1;
    IfdFeatVect          Fv2;
    Real                 Threshold;
    Integer              ColorContentWt;
    Integer              ShapeContentWt;
    Integer              TextureContentWt;
    Integer              BrightnessStructureWt;
    Integer              ColorStructureWt;
    Integer              AspectRatioWt;
    Real                 Score;
```

Fv1, Fv2	Specifies feature vectors to be compared.
Threshold	Specifies for resemblances a number between 0.0 and 1.0.
ColorContentWt	Specifies for color content a number between 0 and 100.
ShapeContentWt	Specifies for shape content a number between 0 and 100.
TextureContentWt	Specifies for texture content a number between 0 and 100.

BrightnessStructureWt	Specifies for brightness structure a number between 0 and 100.
AspectRatioWt	Specifies for aspect ratio a number between 0 and 100.
Score	Specifies a similarity measure of two feature vectors.

Usage

The **Resembles()** function returns a Boolean value indicating whether the resemblance score of two different IfdFeatVect values exceeds a user-defined threshold.

The weight factors are relative to each other and only determine a feature's relative contribution to the final score.

Returns

On success, returns a value if the score is greater than or equal to the threshold. If the score is below the threshold, it returns a false value. The score is subsequently set to 0.0.

ScaleImgBy()

Scales a stored image by the specified scaling factor.

Syntax

```
IfdImgDesc ScaleImgBy(image, factor, quality)
    IfdImgDesc image;
    REAL factor;
    CHAR quality;
```

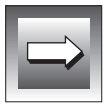
image Specifies the image to be scaled.

factor Specifies the scaling factor (must be greater than 0).

quality Specifies the quality factor: `H` (or `h`) for high-quality, low-speed bilinear interpolation algorithm, or `L` (or `l`) for low-quality, high-speed sampling algorithm.

Usage

The **ScaleImgBy()** function scales an image by a desired factor. To control the quality and speed of the scaling, set the *quality* parameter to `H` or `L`. A value of `H` produces a high-quality resampled image but requires more CPU cycles than the low-quality setting, `L`.



Important: *Images of image type CTAB or HSV are scaled using the low-quality, high-speed algorithm, regardless of the values passed to the function.*

The **ScaleImgBy()** function updates the *pixelwidth* and *pixelheight* fields of the IfdImgDesc data type.

Returns

On success, returns an updated IfdImgDesc object that contains the scaled image. Images are returned as smart large objects regardless of the source image data type.

Example

```
UPDATE gallery
          SET image = ScaleImgBy(image, 1.5, 'H')
          WHERE img_id = 4;
```

This example scales an image by a factor of 1.5, returning an image 50% larger than the original. The `h` setting for the quality parameter specifies a high-quality image produced by the slower scaling algorithm.

ScaleImgTo()

Scales a stored image to a specified pixel width and height.

Syntax

```
IfdImgDesc ScaleImgTo(image, width, height, quality, same_aspect)
    IfdImgDesc image;
    INTEGER width;
    INTEGER height;
    CHAR quality;
    BOOLEAN same_aspect;
```

image Specifies the image to be scaled.

width Specifies the new width, in pixels.

height Specifies the new height, in pixels.

quality Specifies the quality factor: `H` (or `h`) high-quality for low-speed bilinear interpolation algorithm, or `L` (or `l`) for low-quality, high-speed sampling algorithm.

same_aspect Specifies the preserve aspect ratio. Set to `T` (or `t`) for true or `F` (or `f`) for false.

Usage

The **ScaleImgTo()** function scales an image to a desired size. To control the quality and speed of the scaling, set the quality factor to `H` or `L`. A value of `H` produces a high-quality image but requires more CPU cycles than the low-quality setting, `L`.

If you set the *same_aspect* parameter to `T` (true), the **ScaleImgTo()** function preserves the aspect ratio of the original dimensions of the image. The resulting images might be different from the source image.

The **ScaleImgTo()** function updates the *pixelwidth* and *pixelheight* fields of the IfdImgDesc data type.

Returns

On success, returns an IfdImgDesc object that contains the scaled image. Images are returned as smart large objects regardless of the source image data type.

Example

```
UPDATE gallery
          SET image = ScaleImgTo(image, 256, 256, 'L', 'F')
              WHERE img_id = 3;
```

Version()

Returns an integer that is related to the version of the Excalibur Image DataBlade module used to build the IfdFeatVect value.

Syntax

```
Version(FV IFdFeatVect);
```

Fv Specifies an IfdFeatVect vector value.

Usage

The **Version()** function primarily helps with integrity checking and problem diagnosis. For reasons of speed and storage, this number is not the actual release string of the IFV or the VRW version.

Returns

On success, returns an integer encoding the version of the **GetFeatureVector()** function used to extract the specified feature vector. In this release, this value is always equal to 1.

WriteImg()

Writes an image from the database to the location specified by the input IfdLocator value on the server system.

Syntax

```
IfdLocator WriteImg(image, dest_loc, imgformat, imgtype,
overwrite,frame_no)
    IfdImgDesc image;
    IfdLocator dest_loc;
    VARCHAR(25) imgformat;
    VARCHAR(25) imgtype;
    BOOLEAN overwrite;
    INTEGER frame_no;
```

image	Specifies the source image to be written.
dest_loc	Specifies the destination location for the image.
imgformat	Specifies the image file format. Possible values are `AUTO` or one of the formats listed in "Supported Image Formats" on page A-1.
imgtype	Specifies the image file type. Possible values are `AUTO` or one of the image types listed in "Supported Image Types" on page A-2.
overwrite	Specifies the file overwrite option `T` (true) or `F` (false).
frame_no	Specifies the file frame number (`0` for single-frame images).

Usage

The **WriteImg()** function writes an image from an IfdImgDesc row type to the destination specified in the *location* field of an IfdLocator structure stored on the server. See "The IfdLocator Data Type" on page 4-6 for more information.

Set the *imgformat* and *imgtype* parameters to AUTO if you want the destination image to retain the same format as the source image. To change the format of the destination image, set the *imgformat* parameter to the desired image format.

For a list of supported image formats, see "Supported Image Formats" on page A-1.

Set the *imgtype* parameter to AUTO if you want the destination image to retain the same image type as the source image. To change the image type of the destination image, set the *imgtype* parameter to the desired image type.

If an unsupported image type is specified in the *imgtype* or the *imgformat* parameter, the **WriteImg()** function returns an error and does not write to the destination IfdLocator structure.

For a list of supported image types, see "Supported Image Types" on page A-2.

When the overwrite parameter is set to T (true) and the *dest_loc* parameter references an existing file, the **WriteImg()** function overwrites the file. If the format supports multiple frames, the **WriteImg()** function overwrites the frame only. If the specified frame does not exist, the **WriteImg()** function appends the image to the file. If the file exists and the *overwrite* flag is set to F (false), the **WriteImg()** function does not write the file and returns an error. If the file does not exist, the **WriteImg()** function creates it.

The *frame_no* parameter must be 0 for single-frame image formats.

Returns

On success, returns the destination IfdLocator structure with the source image written to the lld_locator data type.

See the *Informix Large Object Locator DataBlade Module User's Guide* for more information on the lld_locator data type.

Example

```
SELECT WriteImg(image,
                        ROW(ROW('ifx_file', NULL::lld_lob,
                                '/tmp/03.jpg')::lld_locator,
                            NULL::lvarchar)::IfdLocator,
                        'AUTO', 'AUTO', "T", 0)
           FROM gallery  WHERE img_id = 3;

SELECT WriteImg(image,
                        ROW(ROW('ifx_file', NULL::lld_lob,
                                '/tmp/02.bmp')::lld_locator,
                            NULL::lvarchar)::IfdLocator,
                        'BMP', 'RGB', "T", 0)
           FROM gallery  WHERE img_id = 4;
```

Appendix

A

Supported Image Formats and Types

Supported Image Formats

External source formats are the image file formats that the Excalibur Image DataBlade module supports, such as GIF or TIFF. If your images are stored in an image format that is available, you can load them directly into an IfdImgDesc column, thus providing your server with access to your data.

The following table summarizes information about the image formats supported by the Excalibur Image DataBlade module.

Value	Format File Extension	Description
BMP	.bmp, .BMP	A format originally used in Windows and OS/2 applications. Not to be confused with the bitmap image type.
GIF	.gif, .GIF	A format that can only store COLORTAB type images. There are patent rules for software that creates GIF files.
JFIF	.jpg, .jpe, .jpeg, .JPG, .JPE, .JPEG	JPEG file interchange format. JPEG uses a lossy compression scheme.
TIFF	.tif, .tiff, .TIF, .TIFF	Tagged image file format. A format that supports multiframe files.

Value	Format File Extension	Description
PDA	.pda, .PDA	Processed document architecture.
PNG	.png, .PNG	Portable Network Graphics. Uses a compression deflation algorithm.
FIT	.fit, .FIT	Sample movie file format specified by Silicon Graphics, Inc.

(2 of 2)

Supported Image Types

The Excalibur Image DataBlade module supports the image types listed in the following table.

Value	Description
BITMAP	Bitmap image type having one bit per pixel. The setting of the bit indicates either black or white. There is no single convention on the interpretation of a set bit.
GRAY	Grayscale image type having 8 bits per pixel, capable of representing 256 shades of gray.
CTAB	Colortable (CTAB) image type having 8 bits per pixel. The value for each pixel is an index in a color table that can contain up to 256 colors.
RGB	Red-green-blue image type having 24 bits per pixel. The first 8 bits indicate the amount of red, the next 8 bits indicate the amount of green, and the last 8 bits indicate the amount of blue.
HSV	Same structure as RGB but the 8-bit segments represent hue, saturation, and value in lieu of red, green, and blue. HSV is supported only with images in EXIMG format.

Appendix

B

Internal Data Structures, Data Types, and Functions

This appendix describes the C structures, data types, and functions supplied by the Excalibur Image DataBlade module for DataBlade module developers. These structures, data types, and functions allow you to:

- read images into your server's shared memory, where you can perform additional image processing on them.
- write images from shared memory to update the IfdImgDesc row in which the image is stored.
- free the memory and the pointer after you have processed your images.

Data Structures and Types

The Excalibur Image DataBlade module supplies the following data structures for DataBlade module developers:

- IfdImage
- IfdIpsImg

It also provides the IfdImagePtr data type.

The IfdImage Data Structure

The IfdImage data structure contains information about the characteristics and canonical representation of an image. This structure contains pointers to:

- the image data structure IfdIpsImg.
- the user field reserved for use by DataBlade module developers.

The type definition for the IfdImage data structure is as follows:

```
typedef struct
{
    IfdIpsImg *IfdMemImg;
    void *user;
}IFDx;

#define IfdImage IFDx;
```

The *user* field is designed for DataBlade module developers using the data types and routines of the Excalibur Image DataBlade module. The Excalibur Image DataBlade module does not use this field. For example, a DataBlade module developer can use this field as an alternative data structure representation by attaching the output of a translation function, such as the **ReadToMem()** function.

The IfdIpsImg structure is described in the following section.

The IfdIpsImg Data Structure

The IfdIpsImg structure is used to represent the characteristics of the images stored and manipulated by the routines supplied by the Excalibur Image DataBlade module. By default, functions read images into an IfdIpsImg structure.

You can pass in the name of a user-defined routine that accepts the IfdImage structure as the only input parameter. The **ReadToMem()** function executes this user-defined routine after creating the IfdImage data type. The pointer to the structure that the conversion user-defined routine creates is the output from the **ReadToMem()** function, which is attached to the user component of the IfdImage structure.

The type definition of the IfdIpsImg data structure is as follows:

```
/* in-memory image structure */
typedef struct
        {
mi_integer      width;
        mi_integer      height;
        mi_pointer      data;
        mi_pointer      user;
        mi_integer      type;
        mi_integer      reserved0;
        mi_integer      datasz;
        mi_integer      nctab;
        uint1           *ctab;

        mi_double_precision xdpi;
        mi_double_precision ydpi;

        mi_integer      reserved_nums[8];
        mi_pointer      reserved_pointers[8];
} IFDimage;

#define IfdIpsImg IFDimage
```

The following list describes the fields of the IfdIpsImg structure.

width	The *x*-dimension of the image, in pixels.
height	The *y*-dimension of the image, in pixels.
data	Pointer to pixel data. Currently, the data is organized in units of unsigned bytes (8-bit integer). Each row is padded to the nearest byte boundary, and the number of bytes per row in an image is consistent. The data must be in "most significant bit order."
user	A member reserved for the application programmer. IPS does not access this member. However, it is not guaranteed to be carried through IPS functions: the application programmer must ensure proper setting in the destination image after any IPS function. Also, freeing an image invalidates the memory of user.

type

This member specifies most of the information necessary for accessing bits and bytes in data. Every row starts on a byte boundary.

If *IFD_BINARY*, then every byte contains 8 pixels, starting with the most significant bit. Bit value `0` is white; bit value `1` is black. The value of a padding bit is unspecified.

If *IFD_GRAY*, then every byte contains 1 pixel. Byte value `0` is black, and byte value `255` is white.

If *IFD_CTAB*, then every byte contains an index into `ctab`, which defines the color of the pixel. The index is not allowed to access elements beyond `nctab`.

If *IFD_RGB/IFD_HSV*, then three consecutive bytes represent the red / hue, green / saturation, and blue / value component of a pixel, each byte specifying a strength of `0` to `255` for its component (except for hue, which can be from `0` to `180`).

datasz

Size in bytes of the buffer pointed to by data. This member must always be set and may be larger than the minimal amount dictated by the image's type, width, and height (uncompressing images can cause such a scenario).

nctab	The number of entries in `ctab`. Set to `0` if `ctab==NULL`.
ctab	Pointer to a buffer containing a color table. If null, the image has no color table; otherwise, the image type is `CTAB`. The maximum allowable number of entries in the table is 256. Each color table entry uses three unsigned bytes: the first byte for the red component, the second for the green component, and the third for the blue component of the color. The length of the color table in bytes is three times its number of entries. For example, `ctab[17]` contains the blue component of the color at color table entry 5.
xdpi, ydpi	X and Y resolution of the image, measured in dots per inch. These values are not currently used by `IFD`, but they are loaded from and saved to files.

The following table defines the in-memory canonical format used to represent images.

Image Type	Canonical Format Representation
IFD_Binary	1-bit representation
IFD_Gray	8-bit representation
IFD_CTAB	8-bit representation
IFD_RGB	24-bit representation
IFD_HSV	24-bit representation (supported in EXIMG format)

The following table defines the pixel code arrangements based on the image type for uncompressed data.

IfdIspImg Type	Bits Per Pixel Code	Bytes Per Row	Pixel Code Arrangement	Photometric
IFD_BINARY	1	(width+7)/8	Order is from most to least significant bit within a byte.	0 is white, 1 is black.
IFD_GRAY	8	width	The entire byte is the pixel code.	0 is black, 255 is white.
IFD_CTAB	8	width	The entire byte is an index into *ctab*. The index is not allowed to be bigger than *nctab*.	The color looked up in *ctab*.
IFD_RGB	8,8,8	3*width	One byte each for red, green, and blue, in that order.	The color read in data.
IFD_HSV	8,8,8	3*width	One byte each for hue, saturation, and value, in that order.	The color read in data.

The IfdImagePtr Data Type

The IfdImagePtr data type is a distinct pointer used by the Excalibur Image DataBlade module to point to the IfdImage structure. This type of pointer is required by the following functions:

- **FreeImagePtr()** function (as input)
- **ReadToMem()** function (as output)
- **WriteFromMem()** function (as input)
- **WriteLocFromMem()** function (as input)

The IfdImagePtr data type can also be used by other image-based DataBlade modules to reference the IfdImage structure.

The type definition for the IfdImagePtr data type is as follows:

```
CREATE DISTINCT TYPE IfdImagePtr AS POINTER;
```

Functions

This section contains reference pages for the functions supplied by the Excalibur Image DataBlade module for DataBlade module developers. These functions operate on the C structures described in "Data Structures and Types" on page B-1.

You can use the I/O routines to read and write images from IfdImgDesc rows into the structures, where you can perform further image processing on them. When you are finished with the processing, you can deallocate memory for the structures.

Function	Description	Page Number
FreeImagePtr()	Frees an IfdImagePtr pointer	page B-8
ReadToMem()	Reads an image from the source location into an IfdImage structure in shared memory	page B-9
WriteFromMem()	Writes an image in shared memory to a destination IfdImgDesc	page B-11
WriteLocFromMem()	Writes an image in shared memory to a destination IfdLocator	page B-13

FreeImagePtr()

Frees an IfdImagePtr pointer.

Syntax

```
Void FreeImagePtr(imgstruct)
    IfdImagePtr imgstruc;
```

imgstruct Specifies the pointer structure you want to free.

Usage

The **FreeImagePtr()** function deallocates the memory allocated by the **ReadtoMem()** function for the IfdImage structure.

Returns

On success, returns `VOID`.

ReadToMem()

Reads an image from an IfdImgDesc row type into an IfdImage structure in shared memory.

Syntax

```
IfdImgPtr ReadToMem(image, imgformat, frame_no,
translate_function)
    IfdImgDesc image;
    VARCHAR(25) imgformat;
    INTEGER frame_no;
    VARCHAR(255) translate_function;
```

image	Specifies the IfdImgDesc row type.
imgformat	Specifies the image file format. Possible values are `AUTO`, `EXIMG`, or one of the formats listed in "Supported Image Formats" on page A-1.
frame_no	Specifies the file frame number (`0` for single-frame images).
translate_function	Specifies the optional structure reformat function specifier.

Usage

The **ReadToMem()** function reads an input image of row type IfdImgDesc into the IfdImage structure in shared memory so that further image processing can take place. The **ReadToMem()** function eliminates the need for individual image-based DataBlade modules to provide their own image I/O functions. Memory for the IfdImage structure, and the data and image it references, is allocated with PER_COMMAND memory duration.

The *translate_function* parameter is an optional user-defined routine that is written and registered in the database by the user. When the *translate_function* user-defined routine is defined by the user, it converts the IfdImage structure into a user-defined structure representing the in-memory image representation. If the *translate_function* parameter is specified (that is, not NULL), the **ReadToMem()** function composes a function signature as follows:

```
translate_function(IfdImagePtr)
```

The **ReadToMem()** function creates the IfdImagePtr structure and passes it to the *translate_function* user-defined routine. The *translate_function* user-defined routine returns the pointer to the newly created structure in the user member of the IfdImage structure. The **ReadToMem()** function returns the IfdImagePtr structure.

Tip: *Use the* ***FreeImagePtr()*** *function to free memory for the IfdImage structure created by the* ***ReadToMem()*** *function.*

Returns

On success, returns a pointer (IfdImagePtr) to the IfdImage structure. Applications can cast the IfdImagePtr structure to the IfdImage structure.

WriteFromMem()

Writes an image referenced by an IfdImagePtr pointer in shared memory to a destination IfdImgDesc structure.

Syntax

```
IfdImgDesc WriteFromMem(image, imgformat, frame_no,
dest_locator,)
    IfdImagePtr image;
    VARCHAR(25) imgformat;
    INTEGER frame_no;
    IfdLocator dest_locator;
```

image	Specifies the IfdImagePtr pointer.
imgformat	Specifies the source image file format. Possible values are `AUTO`, `EXIMG`, or one of the formats listed in "Supported Image Formats" on page A-1. EXIMG is supported for Excalibur DataBlade modules only.
frame_no	Specifies the file frame number (`0` for single-frame images).
dest_locator	Specifies the destination for image specified in an IfdLocator structure.

Usage

The **WriteFromMem()** function writes an image located in shared memory and referenced by an IfdImagePtr structure to a destination in the IfdLocator structure. The IfdLocator structure is attached to an IfdImgDesc row type.

The **WriteFromMem()** function is designed so that the Excalibur Image DataBlade module can write an image from shared memory to an IfdLocator structure. The **WriteFromMem()** function creates an IfdImgDesc row type containing the destination IfdLocator structure. The **WriteFromMem()** function eliminates the need for individual image-based DataBlade modules to provide their own image I/O functions.

Returns

On success, returns an IfdImgDesc row whose fields are populated with values corresponding to the characteristics of the image in the shared-memory structure referenced by the IfdImagePtr structure.

WriteLocFromMem()

Writes an image in shared memory to a destination IfdLocator structure.

Syntax

```
IfdLocator WriteLocFromMem(image, imgformat, frame_no,
dest_locator,
    IfdImagePtr image;
    VARCHAR(25) imgformat;
    INTEGER frame_no;
    IfdLocator dest_locator;
```

image	Specifies the IfdImagePtr pointer.
imgformat	Specifies the source image file format. Possible values are AUTO, EXIMG, or one of the formats listed in "Supported Image Formats" on page A-1.
frame_no	Specifies the file frame number (0 for single-frame images).
dest_locator	Specifies the destination for the image.

Usage

The **WriteLocFromMem()** function writes an image located in memory and referenced by an IfdImagePtr structure to a destination IfdLocator that is not attached to an IfdImgDesc structure. Use the **WriteFromMem()** function if the destination image is attached to an IfdImgDesc structure.

The WriteLocFromMem function is designed so that the Excalibur Image DataBlade module routines or other image-based user-defined routines can write an image from shared memory to an IfdLocator structure. The **WriteLocFromMem()** function eliminates the need for individual image-based DataBlade modules to provide their own image I/O functions.

Returns

On success, returns an IfdLocator structure.

Appendix

C

C Code Examples

This appendix contains the C code examples for the functions supplied by the Excalibur Image DataBlade module for DataBlade module developers. See Appendix B, "Internal Data Structures, Data Types, and Functions," for the function reference pages.

Function	Description	Page Number
FreeImagePtr()	Frees an IfdImagePtr pointer	page B-8
ReadToMem()	Reads an image from the source location into an IfdImage structure in shared memory	page B-9
WriteFromMem()	Writes an image in shared memory to a destination IfdImgDesc	page B-11
WriteLocFromMem()	Writes an image in shared memory to a destination IfdLocator	page B-13

These functions operate on the C structures described in Appendix B.

C Code Examples

The following C code contains examples of the user-defined routines for the Excalibur Image DataBlade module:

```
/****************************************************************************
 *
 *                          INFORMIX SOFTWARE, INC.
 *
 *                             PROPRIETARY DATA
 *
 *      THIS DOCUMENT CONTAINS TRADE SECRET DATA WHICH IS THE PROPERTY OF
 *      INFORMIX SOFTWARE, INC.  THIS DOCUMENT IS SUBMITTED TO RECIPIENT IN
 *      CONFIDENCE.  INFORMATION CONTAINED HEREIN MAY NOT BE USED, COPIED OR
 *      DISCLOSED IN WHOLE OR IN PART EXCEPT AS PERMITTED BY WRITTEN AGREEMENT
 *      SIGNED BY AN OFFICER OF INFORMIX SOFTWARE, INC.
 *
 *      THIS MATERIAL IS ALSO COPYRIGHTED AS AN UNPUBLISHED WORK UNDER
 *      SECTIONS 104 AND 408 OF TITLE 17 OF THE UNITED STATES CODE.
 *      UNAUTHORIZED USE, COPYING OR OTHER REPRODUCTION IS PROHIBITED BY LAW.
 *
 *
 *  Title:       ifdudr.c
 *  CCid:        %W% %E% %U%
 *  Created:     10/96
 *  Description: Example UDR to execute Excalibur Image DataBlade UDRs.
 *
 ****************************************************************************/

#include "ifdudr.h"

/****************************************************************************
* Function:             ifde_init_arrays
*
* Purpose:              Function which allocates SAPI memory for the data and
*                       null arrays used to hold the IfdImgDesc column values
*                       as read in by the ifde_get_colvalues function.
*                       Memory is allocated on a PER_ROUTINE basis for both
*                       arrays.
*
* Inputs:               MI_DATUM **data = data array which will contain
*                                         values of IfdImgDesc columns
*                       mi_boolean **null = null array indicating whether
*                                         associated elements of the
*                                         data array are NULL or not
*                                         NULL
*
* Outputs:              IFDE_ERROR = error in function execution
*                       IFDE_OK = successful function execution
*
****************************************************************************/
```

```
mi_integer
ifde_init_arrays(MI_DATUM **data, mi_boolean **null)

{

        /* allocate data array */
        if ((*data = (MI_DATUM *)mi_dalloc(sizeof(MI_DATUM) * IFDIMG_PARAMS,
                IFDE_MEM_ROUTINE)) == NULL)
                return (IFDE_ERROR);

        /* allocate null array */
        if ((*null = (mi_boolean *)mi_dalloc(sizeof(mi_boolean) * IFDIMG_PARAMS,
                IFDE_MEM_ROUTINE)) == NULL)
                {
                mi_free((char *)*data);
                return (IFDE_ERROR);
                }
}

/************************************************************************
* Function:             ifde_get_colvalues
*
* Purpose:              Function which gets the components of the IfdImgDesc
*                       row type passed to UdrExample.  After reading each
*                       IfdImgDesc column, the data and null arrays are
*                       assigned the element which was just read in.  The
*                       associated ImgLocator and params column positions
*                       within the null array are assigned MI_FALSE since
*                       there will be values assigned to the corresponding
*                       positions within the data array in the ifd_example
*                       function.
*
* Inputs:               MI_ROW *rowimage = IfdImgDesc row passed to UdrExample
*                       MI_DATUM **data = data array which will contain
*                                         values of IfdImgDesc columns
*                       mi_boolean **null = null array indicating whether
*                                         associated elements of the
*                                         data array are NULL or not
*                                         NULL
*
* Outputs:              IFDE_ERROR = error in function execution
*                       IFDE_OK = successful function execution
*
************************************************************************/

mi_integer
ifde_get_colvalues (MI_ROW *rowimage, MI_DATUM *data, mi_boolean *null)

{

        MI_DATUM        colval;         /* return value from mi_value */
        mi_integer      retlen;         /* mi_value's return length value */
        mi_integer      colcnt;         /* column counter */
        mi_integer      retvalue;       /* results from mi_value execution */
```

```
        /* read components of IfdImgDesc excluding IfdLocator column */
        for (colcnt=1; colcnt<=NUMIFDCOLS-1; colcnt++)
                {
                /* extract current value from input IfdImgDesc row type */
                retvalue = mi_value(rowimage, colcnt, &colval, &retlen);
                switch(retvalue)
                        {
                        case MI_NORMAL_VALUE:
                                data[colcnt] = (MI_DATUM)colval;
                                null[colcnt] = MI_FALSE;
                                break;
                        case MI_NULL_VALUE:
                                data[colcnt] = NULL;
                                null[colcnt] = MI_TRUE;
                                break;
                        case MI_ROW_VALUE:
                        case MI_COLLECTION_VALUE:
                        default:
                                return (IFDE_ERROR);
                        }
                }

        /* values will be assigned to the IfdLocator and params component
           of IfdImgDesc, therefore; assign MI_FALSE to the corresponding
           index into the null array */
        null[IFDIMG_LOC] = MI_FALSE;
        null[IFDIMG_PARAMS] = MI_FALSE;

        return (IFDE_OK);
}

/*************************************************************************
* Function:             ifde_read_image_to_mem
*
* Purpose:             Function which calls the Excalibur Image Datablade module's
*                        ReadToMem UDR to read an image referenced in an
*                        IfdImgDesc row type into the server's shared memory
*                        for futhar processing.
*
* Inputs:                MI_CONNECTION * conn = server connection
*                        MI_ROW *rowimage = IfdImgDesc row passed to UdrExample
*                        mi_pointer memimage = returned address of IfdImage
*                                              structure upon successful
*                                              execution of this function
*
* Outputs:               IFDE_ERROR = error in function execution
*                        IFDE_OK = successful function execution
*
*************************************************************************/

mi_integer
ifde_read_image_to_mem(MI_CONNECTION *conn, MI_ROW *rowimage,
```

```
        mi_pointer *memptr)
{
        MI_FUNC_DESC    *funcdesc=NULL; /* external UDR descriptor */
        mi_char         *signature;     /* external UDR signature */
        mi_integer      fpstatus = 0;   /* results from fastpath execution */
        mi_integer      frame=0;

        signature ="function readtomem(ifdimgdesc, VARCHAR(25), INTEGER,
VARCHAR(255))";

        /* get function descriptor for ReadToMem UDR */
        if ((funcdesc = mi_routine_get(conn, 0, signature)) == NULL)
                return (IFDE_ERROR);

        /* execute ReadToMem UDR */
        *memptr = (mi_pointer *)mi_routine_exec(conn, funcdesc, &fpstatus,
                rowimage, mi_string_to_lvarchar("AUTO"), frame, NULL);
        if (fpstatus)
               return (IFDE_ERROR);

        mi_routine_end (conn, funcdesc);

        return (IFDE_OK);

}

/*************************************************************************
* Function:             ifde_do_something_to_image
*
* Purpose:              An example function where further image processing
*                       functions could be applied to the image read into
*                       shared memory by the IFD ReadToMem UDR.

* Inputs:               mi_pointer memimage = image located in shared memory
*
* Outputs:              IFDE_ERROR = error in function execution
*                       IFDE_OK = successful function execution
*
*************************************************************************/

mi_integer
ifd_do_something_to_image(mi_pointer memimage)
{

        IfdImage        *memptr=NULL;   /* IfdImage in-memory structure */

        /* cast void memimage to the IfdImage structure for processing */
        memptr = memimage;

        /*

           IMAGE PROCESSING FUNCTIONS FROM HERE ...

           ...
           ...
```

```
        */


        return(IFDE_OK);

}

/************************************************************************
* Function:             ifde_write_image
*
* Purpose:             Function which calls the Excalibur Image Datablade module's
*                       WriteLocFromMem UDR to create an in-memory IfdLocator
*                       row.
*
* Inputs:               MI_CONNECTION * conn = server connection
*                       mi_pointer memimage = image located in shared memory
*                       mi_lvarchar *format = destination image format
*                       mi_integer frame = frame within image to write out
*                       MI_ROW *dest_ifdlocator = destination IfdLocator
*                       MI_ROW **rowreturn = returned row address upon
*                                            successful execution of this
*                                            function
*
* Outputs:              IFDE_ERROR = error in function execution
*                       IFDE_OK = successful function execution
*
************************************************************************/

mi_integer
ifde_write_image(MI_CONNECTION *conn, mi_pointer memimage,
        mi_lvarchar *format, mi_integer frame, MI_ROW *dest_ifdlocator,
        MI_ROW **rowreturn)
{
        MI_FUNC_DESC    *funcdesc=NULL; /* external UDR descriptor */
        mi_char         *signature;     /* external UDR signature */
        mi_integer      fpstatus = 0;   /* results from fastpath execution */

                signature ="function writelocfrommem(ifdimageptr, VARCHAR(25),
INTEGER, ifdlocator)";

        /* get function descriptor for WriteLocFromMem UDR */
        if ((funcdesc = mi_routine_get(conn, 0, signature)) == NULL)
                return (IFDE_ERROR);     /* invalid UDR signature */

        /* execute WriteLobFromMem/WriteImgFromMem UDR */
        *rowreturn = (MI_ROW *)mi_routine_exec(conn, funcdesc,
                &fpstatus, memimage, format, frame, dest_ifdlocator);

        if (fpstatus)
                return (IFDE_ERROR);

        mi_routine_end (conn, funcdesc);

        return (IFDE_OK);
```

```
}

/*************************************************************************
* Function:             ifde_free_image
*
* Purpose:             Function which calls the Excalibur Image Datablade module's
*                       FreeImagePtr UDR to free the in-memory IfdImage.
*
* Inputs:               MI_CONNECTION * conn = server connection
*                       mi_pointer memimage = image located in shared memory
*
* Outputs:              IFDE_ERROR = error in function execution
*                       IFDE_OK = successful function execution
*
*************************************************************************/

mi_integer
ifde_free_image(MI_CONNECTION *conn, mi_pointer memimage)
{
        MI_FUNC_DESC    *funcdesc = 0;  /* external UDR descriptor */
        mi_char         *signature;     /* external UDR signature */
        mi_integer      fpstatus = 0;   /* results from fastpath execution */

        signature = "procedure freeimageptr(ifdimageptr)";

        /* get function descriptor for FreeImagePtr UDR */
        if ((funcdesc = mi_routine_get(conn, 0, signature)) == NULL)
                return (IFDE_ERROR);     /* invalid UDR signature */

        /* execute FreeImagePtr UDR */
        mi_routine_exec(conn, funcdesc, &fpstatus, memimage);
        if (fpstatus)
                return (IFDE_ERROR);

        mi_routine_end (conn, funcdesc);

        return (IFDE_OK);
}

/*************************************************************************
* Function:             ifde_create_row
*
* Purpose:              Creates an in-memory IfdImgDesc row by getting
*                       the typeid for the IfdImgDesc row type, creating
*                       the associated row descriptor, and finally,
*                       creating the row.  The ifd_data array and null array
*                       are passed into this function for the row creation
*                       function, mi_row_create.
*
* Inputs:               MI_CONNECTION * conn = server connection
*                       MI_DATUM *ifd_data = pointer to array containing
*                                            column values for IfdImgDesc
*                       mi_boolean *null = array specifying whether associated
```

```
*                                               ifd_data array components
*                                               contain null elements
*                       MI_ROW **rowimage = address of row pointer which will
*                                               be assigned the returned row
*                                               upon successful execution of
*                                               this function
*
* Outputs:              IFDE_ERROR = error in function execution
*                       IFDE_OK = successful function execution
*
*****************************************************************************/

mi_integer
ifde_create_row(MI_CONNECTION *conn, MI_DATUM *ifd_data, mi_boolean *null,
        MI_ROW **rowimage)
{

        MI_ROW_DESC     *rowdesc;
        MI_TYPEID       *typeid;
        MI_ROW_DESC     **row_desc;

        /* get typeid for the IfdImgDesc row type */
        if ((typeid = mi_typestring_to_id(conn, ROWTYPE)) == NULL)
                return (IFDE_ERROR);

        /* get an MI_ROW_DESC for the IfdImgDesc row type */
        if ((rowdesc = mi_row_desc_create(typeid)) == NULL)
                return (IFDE_ERROR);

        /* create a new row */
        if ((*rowimage = mi_row_create(conn, rowdesc, ifd_data, null)) == NULL)
                return (IFDE_ERROR);

        return (IFDE_OK);
}

/*****************************************************************************
* Function:             ifde_ifd_clean_up_and_exit
*
* Purpose:              Function is called by ifd_example function when
*                       any error is encountered during execution.  This
*                       function frees all memory structures and raises an
*                       MI_SQL exception through mi_db_error_raise with the
*                       appropriate SQLSTATE error code.
*
* Inputs:               MI_CONNECTION * conn = server connection
*                       MI_DATUM *ifd_data = pointer to array containing
*                                               column values for IfdImgDesc
*                       mi_boolean *null = array specifying whether associated
*                                               ifd_data array components
*                                               contain null elements
*                       mi_string *sqlstate = SQLSTATE error code
*                       mi_pointer memimage = image located in shared memory
*
```

```
* Outputs:              void
*
************************************************************************/

void
ifd_clean_up_and_exit(MI_CONNECTION *conn, MI_DATUM *data, mi_boolean *null,
        mi_string *sqlstate, mi_pointer memimage)
{

        if (data != NULL)
                mi_free((char *)data);

        if (null != NULL)
                mi_free((char *)null);

        if (memimage != NULL)
                ifde_free_image(conn, memimage);

        mi_close(conn);
        mi_db_error_raise (NULL, MI_SQL, sqlstate, NULL);

}


/************************************************************************
* UDR Name:             UdrExample
*
* Purpose:              Example code which demonstrates how to use the
*                       ReadToMem, WriteFromMem and FreeImagePtr UDRs
*                       within a UDR. This is a very simple UDR which
*                       does the following:
*
*                       1 - accepts as input, an IfdImgDesc row type
*                           and an lvarchar value to update the params
*                           component of the IfdImgDesc row type
*                       2 - reads the image referenced in the IfdLocator
*                           into server memory, applies a
*                           "do-something-to-image function" (note: it is at
*                            this point where a developer could do their
*                            own API processing on the in-memory image)
*                       3 - updates the IfdLocator with the updated image
*                       4 - updates the params column of the IfdImgDesc
*                           row type with the input lvarchar
*                       5 - creates an IfdImgDesc row type and outputs the
*                           row to be inserted
*
*                       Error handling used in this example UDR is very
*                       simple... errors encountered after each step through
*                       results in a call to the ifd_clean_up_and_exit function
*                       where SAPI memory is deallocated and
*                       mi_db_error_raise is called with the appropriate
*                       SQLSTATE error code.
*
* Sample Call:          INSERT INTO ifdtable2
```

```
*                               SELECT UdrExample(picture, 'test')
*                               FROM ifdtable1
*                               WHERE id = 1;
*
* Inputs:               picture = IfdImgDesc column referenced in
*                                 table ifdtable2
*                       somevalue = an lvarchar that will be assigned to
*                                 the params column of the input IfdImgDesc
*                                 row type

* Outputs:              MI_ROW * containing the new IfdImgDesc row
*
*
**************************************************************************/

MI_ROW *
ifd_example (MI_ROW *rowimage, mi_lvarchar *somevalue)
{
        MI_CONNECTION   *conn=NULL;     /* server connection */
        MI_ROW          *destimage=NULL;/* destination IfdImgDesc to create */
        MI_ROW          *destlobloc=NULL;/* destination IfdLocator */
        MI_DATUM        colval=NULL;    /* results from mi_value */
        MI_DATUM        *ifd_data;      /* data for IfdImgDesc row type */
        mi_boolean      *null;          /* null data for IfdImgDesc row type */
        mi_pointer      memimage=NULL;  /* public in-memory image structure */
        mi_integer      retlen=0;       /* mi_value's return value length */
        mi_integer      status=0;       /* status of function execution  */

        /* make default connection to server */
        if((conn = mi_open(NULL, NULL, NULL)) == NULL)
                mi_db_error_raise (NULL, MI_SQL, IFDE_CONN_ERROR, NULL);

        /* initialize IFD_IO structure with current row type values */
        if (ifde_init_arrays(&ifd_data, &null) == IFDE_ERROR)
                {
                mi_close(conn);
                mi_db_error_raise (NULL, MI_SQL, IFDE_MEM_ALLOC_ERROR, NULL);
                }
        if (ifde_get_colvalues(rowimage, ifd_data, null)
                == IFDE_ERROR)
                ifd_clean_up_and_exit(conn, ifd_data, null, IFDE_COLINIT_ERROR,
                        NULL);

        /* get destination IfdLocator into colval */
        if (mi_value(rowimage, IFDIMG_LOC, &colval, &retlen)
                != MI_ROW_VALUE)
                ifd_clean_up_and_exit(conn, ifd_data, null, IFDE_IFDLOC_ERROR,
                        NULL);

        /* read input mi_row type into its in-memory canonical format */
        if (ifde_read_image_to_mem(conn, rowimage, &memimage)
                == IFDE_ERROR)
                /* memory allocated for destlobloc by mi_value is auto-freed*/
                ifd_clean_up_and_exit(conn, ifd_data, null,
```

```
                        IFDE_READTOMEM_ERROR, NULL);

        /* apply image processing functions to in-memory image */
        if (ifd_do_something_to_image(memimage) == IFDE_ERROR)
                ifd_clean_up_and_exit(conn, ifd_data, null, IFDE_APIIMG_ERROR,
                        memimage);

        /* write IfdImage in-memory image to destination IfdLocator */
        if (ifde_write_image(conn, memimage, mi_string_to_lvarchar("AUTO"),
                DFLTFRAME, (MI_ROW *)colval, &destlobloc) == IFDE_ERROR)
                        ifd_clean_up_and_exit(conn, ifd_data, null,
                                IFDE_APIIMG_ERROR, memimage);

        /* update ifd_data and null arrays with new values */
        ifd_data[IFDIMG_LOC] = (MI_DATUM)destlobloc;
        ifd_data[IFDIMG_PARAMS] = (MI_DATUM)somevalue;

        /* create new row with ifd_data and null */
        if (ifde_create_row(conn, ifd_data, null, &destimage) == IFDE_ERROR)
                ifd_clean_up_and_exit(conn, ifd_data, null,
                        IFDE_WRITEFROMMEM_ERROR, memimage);

        /* clean up and return new IfdImgDesc row */
        mi_free((char *)ifd_data);
        mi_free((char *)null);
        ifde_free_image(conn, memimage);
        mi_close (conn);

        return (destimage);
}
```

Glossary

aspect ratio A measure of the ratio of the width to the height of an image.

brightness structure A measure of the brightness at each point in an image.

color content A measure of the colors in an image, including their relative proportions.

color structure A measure of the hue, saturation, and brightness at each point in an image.

feature extractor function A function that decodes the important attributes of an object into feature vectors. The Excalibur Image DataBlade module provides color content and structure, shape content, brightness content, texture content, and aspect ratio feature extractor functions.

feature extractor parameter A parameter specified by the user and passed to the feature extractor function to set the characteristics of feature vectors.

feature vector An array of bits representing the presence or absence of specific features. Feature vectors are created by a feature extractor function.

score A measure of the similarity of two feature vectors.

search image The image used to search a database for matching images. The search image is passed as a parameter of the **Resembles()** operator function.

shape content A measure of the relative orientation, curvature, and contrast of lines in an image.

statement local variable (SLV) A variable whose scope is limited to the statement in which it is used.

texture content A measure, on a small scale, of the flow and roughness of an image.

threshold A number between 0.0 and 1.0 used to set the resemblances of feature vectors.

Index

R

S

W