Excalibur Image DataBlade Module

User's Guide

Version 1.1
February 1998
Part No. 000-4362

Published by INFORMIX® Press

Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025-1032

ACKNOWLEDGMENTS

Documentation Team: Clyanne Tuuri, Tom DeMott, Dorothy Moore, Claire Mosher, Howard Resnick

Contributors: Jackie Ryan, Armand Wilson

Table of Contents

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 this manual.

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 Dynamic Server with UD Option or, simply, 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

The Excalibur Image DataBlade module is made up of three components: Image Processing, Image Foundation, and Features. An overview chapter describes how these components work together to support the storage and retrieval of images using the Dynamic Server with UD Option.

This manual includes the following chapters:

- This introduction provides an overview of the manual, describes the documentation conventions used, and explains the generic style of this documentation.
- Chapter 1, "Overview of the Excalibur Image DataBlade Module," provides an overview of the product functionality and describes the Image Foundation, Image Processing, and Features components.

- Chapter 2, "Data Types," documents the data types and structures defined in the Excalibur Image DataBlade module.
- Chapter 3, "Image I/O and Manipulation," provides reference pages for the image manipulation and I/O functions.
- Chapter 4, "Feature-Based Image Retrieval," describes the color, shape (gestalt), and texture feature extractors; index creation; and SQL syntax for image retrieval.
- Appendix A, "Supported Image Formats and Types," documents the image types provided by the Excalibur Image DataBlade module.
- Appendix B, "Feature Extractor Functions," provides reference pages for the feature extractor functions described in Chapter 4.
- Appendix C, "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 D, "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 the 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 **frnet** access method and data types supplied by the Excalibur Image DataBlade module.

Software Dependencies

To use this manual, you must be using the Universal Data Option. Check your release notes to determine which versions of the 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 LOB 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.
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.

(1 of 2)

Convention	Meaning
`monospace`	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
♦	This symbol indicates the end of product- or platform-specific information.

(2 of 2)

Tip: *When you are instructed to "enter" characters or to "execute" a command, immediately press* RETURN *after the entry. When you are instructed to "type" the text or to "press" other keys, no* RETURN *is required.*

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 Image Foundation, Image Processing, and Features components work together 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:

- *LOB Locator DataBlade Module Programmer's Guide*
- *DataBlade Developers Kit User's Guide*
- *BladeManager User's 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
Documentation notes	Describe features not covered in the manuals or modified since publication.
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.

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

Informix Welcomes Your Comments

Please tell us 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. Please include the following information:

- The name and version of the manual you are using
- Any comments 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

Or, send a facsimile to Technical Publications at:

650-926-6571

We appreciate your feedback.

Chapter

1

Overview of the Excalibur Image DataBlade Module

This chapter provides an overview of the Excalibur Image DataBlade module components and how they support 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 indexed by content attributes in the database, allowing you to select an image and search the index to locate similar images.

The image technology contained in the Excalibur Image DataBlade module is based on neural network search techniques. The Excalibur Image DataBlade module allows you to perform content-based searches of groups of images by indexing the binary patterns in digital information.

Prerequisites

The Excalibur Image DataBlade module requires the following software:

- Informix Dynamic Server with Universal Data Option.
- LOB 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 an indexed table, a *feature extractor function* records the characteristics of the image in an array of bits called a *feature vector*. The feature vector (not the image itself) is indexed, providing content-based access to the images in the database table.

- search for matching and similar images in the database.

  When images are retrieved, a feature extractor creates a feature vector for a target *search image*. The index of the database table is searched for similar vectors, the image associated with each similar vector is retrieved, and a ranked list of matching images is returned.

The following diagram illustrates the functionality of the Excalibur Image DataBlade module.

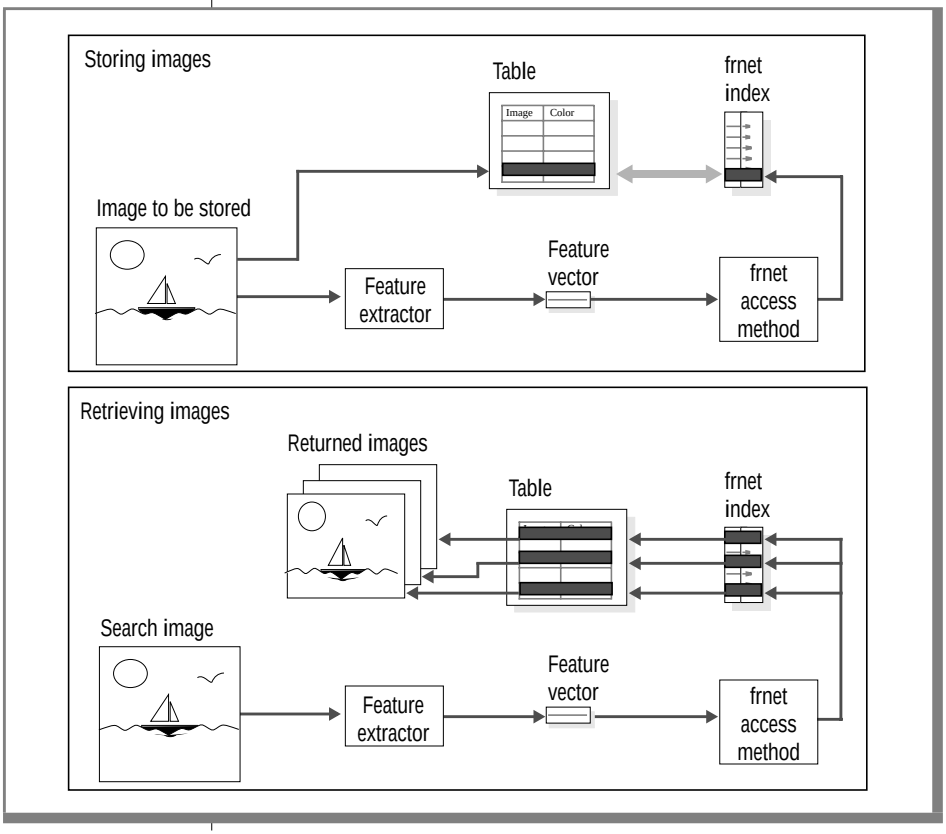


Figure 1-1
Storing and Retrieving Images

Architecture

The Excalibur Image DataBlade module is made up of the following components:

- Image Processing
- Image Foundation
- Features

These components interact to provide the storage, retrieval, and search capabilities of the Excalibur Image DataBlade module. For example, the Image Processing component uses the image manipulation and I/O routines of the Image Foundation component and the **frnet** access method of the Features component to store and retrieve images.

The content of the Excalibur Image DataBlade module components is shown in the following diagram.

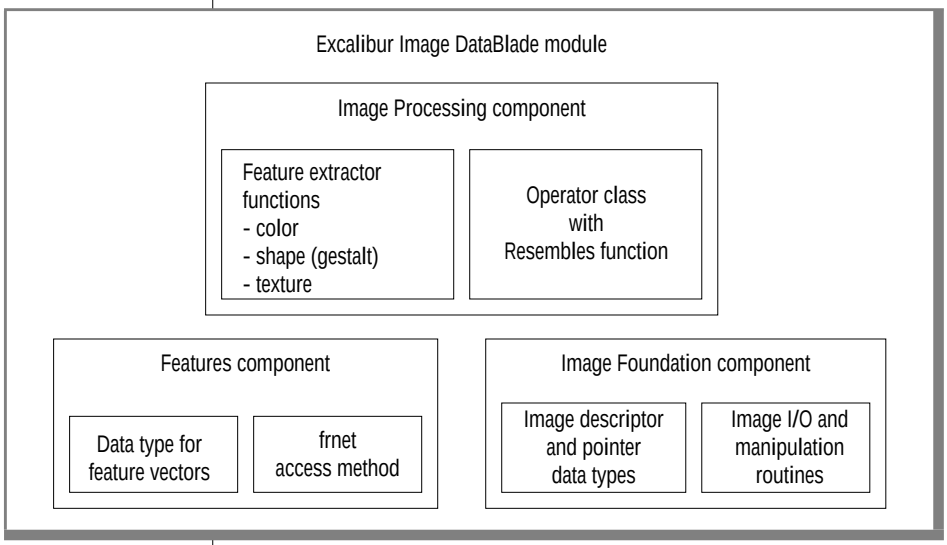


Figure 1-2
Architecture

The following sections describe each component of the Excalibur Image DataBlade module.

Image Processing Component

The Image Processing component of the Excalibur Image DataBlade module lets you:

- retrieve images based on image features such as color, texture, and shape.
- search for similar or matching images in the database.

The Image Processing component includes:

- **Feature extractor functions.** These functions create feature vectors that are indexed by the Features component of the DataBlade module. The feature extractors functions are as follows:
  - The **IpdColorImageFE** function extracts color features of an image.
  - The **IpdTextureImageFE** function extracts texture features of an image.
  - The **IpdGestaltImageFE** function extracts the shape, or *gestalt*, features from an image, using either a unipole or dipole algorithm.

  To extract the features in a table of stored images, you specify one of the feature extractor functions in the USING clause of an SQL CREATE INDEX statement. The feature extractor function creates a separate feature vector for every image stored in the table.

  For more information about the feature extractors, see "Feature Extractor Functions" on page 4-6.
- **IfdImgDesc_ops operator class.** The **IfdImgDesc_ops** operator class supplies the **Resembles** function for the **frnet** access method.
- **Resembles function.** The **Resembles** function is a part of the **IfdImgDesc_ops** operator class. To retrieve images from the database, call the **Resembles** function in the WHERE clause of an SQL SELECT statement. This SQL query passes a search image (that is, the image to be matched) as one of the **Resembles** function parameters.

  For more information on the **Resembles** function, see "Using the Resembles Function" on page 4-18.

Image Foundation Component

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

Using the features of the Image Foundation component, 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.

Features Component

The Features component of the Excalibur Image DataBlade module increases the efficiency of image retrieval by indexing your images based on image content. The Features component uses output from and provides input to the Image Processing component.

The Features component includes the following features:

- The **frnet** index stores the characteristics of the image as feature vectors. The **frnet** index is created by the feature extractors of the Image Processing component of the Excalibur Image DataBlade module.
- The **frnet** access method is the secondary access method used by the Excalibur Image DataBlade module to retrieve images from the database. The **frnet** access method calls the **Resembles** function, which is provided by the Image Processing component of the Excalibur Image DataBlade module.

For more information on the **frnet** index and access method, see Chapter 4, "Feature-Based Image Retrieval."

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 feature-based index
- Inserting images into the database
- Searching for images in the database

The following subsections provide a brief overview of storing and retrieving 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 imgtable (img_id int, image ifdimgdesc);
```

The Excalibur Image DataBlade module supplies this data type along with routines for manipulating your images. See Chapter 2, "Data Types," for more information.

Tip: *Use a browser to display images in the database.*

Creating a Feature-Based Index

To create a feature-based index on the column of data type IfdImgDesc, use the following SQL statement:

```
CREATE INDEX gc_image on imgtable (image IfdImgDesc_ops)
USING frnet(extractor=
    'IpdColorImageFE(RETINA_WIDTH=32,  RETINA_HEIGHT=32)')
in sbspace;
```

The **frnet** index is stored in an sbspace. The sbspace is any smart large object space created for use by the index. You must set a default sbspace in the ONCONFIG configuration file to:

```
SBSPACENAME     sbspace
```

For a full discussion of database server configuration requirements, see your server's administrator's guide.

For more information on the **frnet** access method, which allows you to create the **frnet** index, and the feature extractor **IpdColorImageFE** function specified in the USING clause, see Chapter 4, "Feature-Based Image Retrieval."

Inserting Images into the Database

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

```
INSERT INTO imgtable
VALUES (1,
    row(row(row('ifx_file', NULL::LLD_Lob,
        '/tmp/october.jpg'),
    NULL::lvarchar),
    NULL::lvarchar, NULL::lvarchar, NULL::integer,
    NULL::integer, NULL::lvarchar)::ifdimgdesc);

INSERT INTO imgtable
VALUES (2,
    row(row(row('ifx_file', NULL::LLD_Lob,
        '/tmp/lenncp.tif'), NULL::lvarchar),
        NULL::lvarchar, NULL::lvarchar, NULL::integer,
        NULL::integer, NULL::lvarchar)::ifdimgdesc);
```

Images are indexed as they are inserted into an indexed table.

Searching for Images in the Database

To search the *indexed 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 imgtable
WHERE
Resembles(image,
row(
'IpdColorImageFE(RETINA_WIDTH=32, RETINA_HEIGHT=32)',
row(row(row('IFX_FILE', NULL::LLD_Lob,
    '/tmp/lenncp.tif'),
    NULL::lvarchar), '', '', 0, 0, '')),
    rank #REAL);
```

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

Note that the null- and zero-value parameters in the previous example could be replaced by other values without affecting the results. The initial settings of these parameters do not matter.

The **Resembles** function is part of the **IfdImgDesc_ops** operator class, supplied by the Image Processing component. See Chapter 4, “Feature-Based Image Retrieval,” for more information.

Chapter

2

Data Types

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 LOB Locator DataBlade module.

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 LOB 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 LOB Locator DataBlade module. The *location* field is an IfdLocator data type. This data type is documented in the "The IfdLocator Data Type" section in this chapter.

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-3.

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 Image Foundation component.

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)
);
```

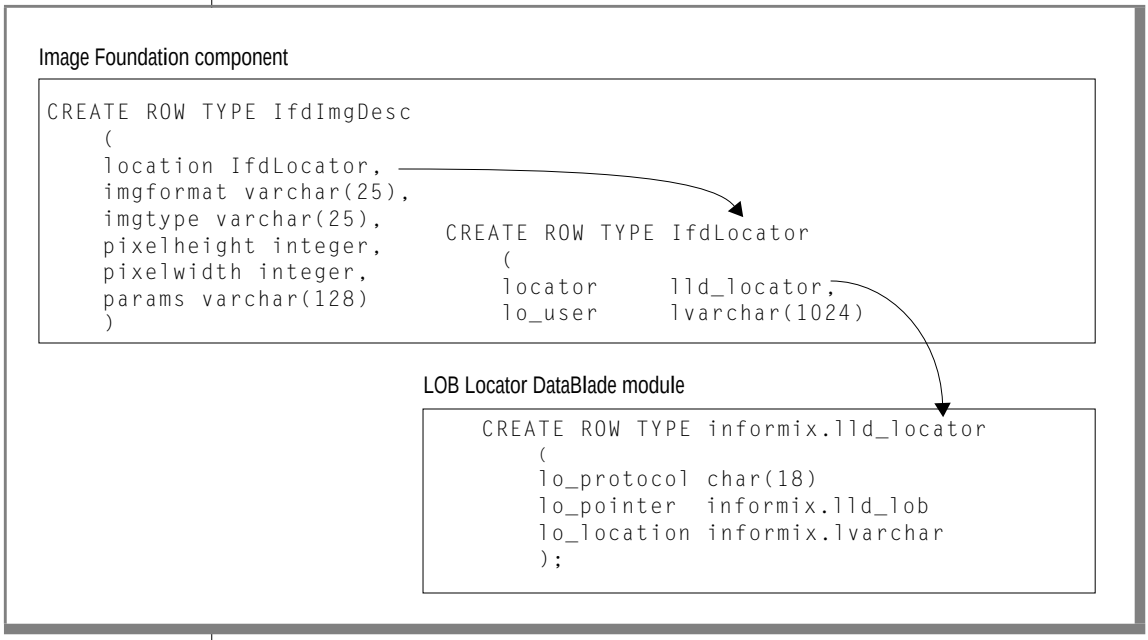
The following list describes the fields of the IfdLocator data type:

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

lo_user Is available for users. The Foundation component of 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 2-1
Relationship of Image Foundation and LOB Locator Data Types



See the *LOB Locator DataBlade Module Programmer's Guide* for more information on the lld_locator data type.

Chapter

3

Image I/O and Manipulation

This chapter provides reference information on the 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. The examples on these reference pages use the image table and index created in previous chapters of this guide. 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.

ScaleImgBy	Scales an image by a specified factor	page 3-18
ScaleImgTo	Scales an image to a specified size	page 3-20
WriteImg	Writes an image from an IfdImgDesc row to an IfdLocator destination	page 3-22

Tip: *The Image Foundation component 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.*

ConvertImgFormat

Converts an image from one format to another.

Syntax

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

or

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

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.

compression Specifies the compression scheme. See "ImgCompression" on page 3-9 for a list of available compression schemes.

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.

For compression parameter values, see “ImgCompression” on page 3-9. In addition to the compression scheme strings described in the **ImgCompression** function, the following string can also be used.

Compression Scheme	Image Type	Definition
JPEG[:factor]	JFIF	The optional value, [:factor], represents the size and quality trade-off value (the JPEG compression algorithm default is 75).

Returns

On success, returns an IfdImgDesc row containing the image in the converted format.

Example

```
UPDATE imgtable
SET image = ConvertImgFormat(image, 'tiff')
WHERE img_id = 1;
```

ConvertImgType

Converts from one image type to another.

Syntax

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

or

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

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-3.
compression	Specifies the compression scheme. See "ImgCompression" on page 3-9 for a list of available compression schemes.

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.

For compression parameter values, see "ImgCompression" on page 3-9. In addition to the compression scheme strings described in the **ImgCompression** function, the following string can also be used.

Compression Scheme	Image Type	Definition
JPEG[:factor]	JFIF	The optional value, [:factor], represents the size and quality trade-off value (the JPEG compression algorithm default is 75).

Returns

On success, returns an IfdImgDesc structure containing the converted image type.

Example

```
UPDATE imgtable
SET image = ConvertImgType(image, 'GRAY')
WHERE img_id = 1;
```

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

(1 of 2)

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

(2 of 2)

Returns

On success, returns a VARCHAR value indicating the compression scheme of the specified image.

Examples

```
SELECT ImgCompression(image) from imgtable;
```

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 indicating the format of the specified image.

Example

```
SELECT ImgFormat(image) from imgtable;
```

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.

Example

```
SELECT ImgType(image) from imgtable;
```

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-3.

Example

```
SELECT ImgType(image) from imgtable;
```

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.

Example

```
SELECT ImgWidth(image) from imgtable;
```

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;
```

or

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

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-3.
frame_no	Specifies the file frame number (0 for single-frame images).
compression	Specifies the compression scheme. See “ImgCompression” on page 3-9 for a list of available compression schemes.

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 Image Foundation component 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 2-6. For more information on setting values for the lld_locator data type, see the *LOB Locator DataBlade Module Programmer'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* in an IFX_BLOB 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.

For compression parameter values, see "ImgCompression" on page 3-9. In addition to the compression scheme strings described in the **ImgCompression** function, the following string can also be used.

Compression Scheme	Image Type	Definition
JPEG[:factor]	JFIF	The optional value, [:factor], represents the size and quality trade-off value (the JPEG compression algorithm default is 75).

Returns

On success, returns an image of data type IfdImgDesc.

Example

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

```
INSERT INTO imgtable
VALUES
(3, ReadImg(row(row
('IFX_FILE', NULL::LLD_Lob, '/tmp/informix.tif'),
NULL::lvarchar)::ifdlocator,
row(row('IFX_FILE', NULL::LLD_Lob,'/tmp/informix1.tif'),
NULL::lvarchar)::ifdlocator, 'AUTO', 'AUTO', 0));
```

ScaleImgBy

Scales a stored image by the specified scaling factor.

Syntax

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

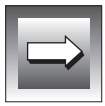
image Specifies the image to be scaled.

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 row type containing the scaled image.

Example

```
UPDATE imgtable
SET image = ScaleImgBy(image, 1.5, 'h')
WHERE img_id = 1;
```

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`), true, or `F` (or `f`), 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 row type containing the scaled image.

Example

```
UPDATE imgtable
SET image = ScaleImgTo(image, 256, 256, 'L', 'f')
WHERE img_id= 1;
```

WriteImg

Writes an image from the database to the location specified by the input IfdLocator 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;
```

or

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

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-3.

overwrite	Specifies the file overwrite option `T` (true) or `F` (false).
frame_no	Specifies the file frame number (`0` for single-frame images).
compression	Specifies the compression scheme. See "ImgCompression" on page 3-9 for a list of available compression schemes.

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 2-6 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-3.

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.

For compression parameter values, see "ImgCompression" on page 3-9. In addition to the compression scheme strings described in the **ImgCompression** function, the following string can also be used.

Compression Scheme	Image Type	Definition
JPEG[:factor]	JFIF	The optional value, [:factor], represents the size and quality trade-off value (the JPEG compression algorithm default is 75).

Returns

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

See the *LOB Locator DataBlade Module Programmer's Guide* for more information on the lld_locator data type.

Example

```
SELECT WriteImg(image,
    row(row('IFX_FILE', NULL::LLD_Lob, '/tmp/tmpimg.png'),
    NULL::lvarchar)::ifdlocator,
    'PNG', 'CTAB', "T", 0)
FROM imgtable
WHERE img_id = 1;
```

Chapter

4

Feature-Based Image Retrieval

This chapter shows how the Excalibur Image DataBlade module uses features to search for and retrieve objects stored in binary format according to their color, shape, and texture.

What Is Feature-Based Retrieval?

The Excalibur Image DataBlade module provides the **frnet** access method, which allows you to perform content-based searches on groups of binary objects using a specialized index called an **frnet** index. The **frnet** access method is modeled on the way biological systems use neural networks to process information.

The Excalibur Image DataBlade module also provides *feature extractor functions* to decode the important features of your objects. A feature extractor function records the presence or absence of a feature in a string of bits called a *feature vector*. Feature vectors can be indexed by the **frnet** access method. The Excalibur Image DataBlade module also provides a data type (FrsFeatVect) for storing feature vectors.

Extracting Features

A *feature extractor* is a function that decodes important attributes of an image into an array of bits called 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 feature vectors to index stored images and to retrieve images from a database by searching the index for matching images. Feature vectors provide the following advantages:

- They allow you to retrieve images based on their content.

- They require less storage than the actual image data.
- They can be used in a neural network search.

Additionally, the process of creating a feature vector provides a method of normalizing an image by removing those features that are not important to a particular application.

The feature vectors created by feature extractor functions are of the data type FrsFeatVect, a distinct BLOB data type. The Excalibur Image DataBlade module uses the feature vectors stored in the FrsFeatVect structure as keys when it creates an **frnet** index. Feature extractor functions are not called directly, but are invoked in the SQL SELECT or INSERT statements.

For more information on the FrsFeatVect data type and the **frnet** index, see "Creating Indexes" on page 4-8.

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 containing 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. This process is illustrated in Figure 4-1.

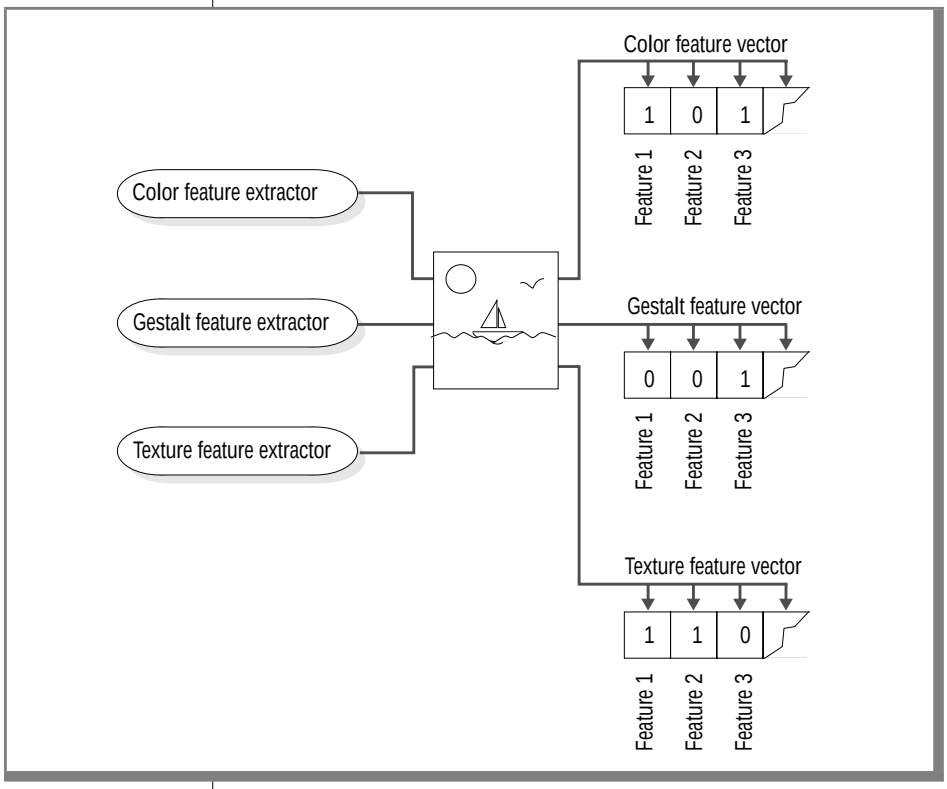


Figure 4-1
Extracting Features into Feature Vectors

The FrsFeatVect Data Type

The FrsFeatVect data type is a distinct BLOB data type used as an output data type by the Image Processing component and as an input data type by the Features component.

The FrsFeatVect data type is designed to record the features of an object in a feature vector created by a feature extractor function. These features are extracted when you create an **frnet** index. You should not attempt to store image data in the FrsFeatVect data type directly.

The FrsFeatVect data type is created using the following SQL statement:

```
CREATE DISTINCT TYPE FrsFeatVect AS BLOB;
```

Feature Extractor Functions

The following feature extractor functions are provided by the Excalibur Image DataBlade module:

- The **IpdColorImageFE** function extracts features from an image based on its color content.
- The **IpdTextureImageFE** function extracts features from an image based on its texture content.
- The **IpdGestaltImageFE** function extracts the shape or the outline of an image based on the overall features contained in the image, using either a unipole or a dipole algorithm.

  The name *gestalt* refers to an image or pattern that is unified as a *whole* so that its properties cannot be derived from its parts. The **IpdGestaltImageFE** function works most effectively with a search image that—on the whole—resembles the indexed images.

For complete reference manual pages for the feature extractor functions, see Appendix B.

Using Feature Extractor Functions

You do not call feature extractor functions directly; rather, the database server calls them when it executes an SQL statement.

Feature extractor functions can be specified in the following SQL statements:

- The USING clause of the CREATE INDEX statement
- The WHERE clause of a SELECT statement

To specify a feature extractor function, you need to know:

- the syntax of the feature extractor function and its parameters.
- the syntax for the *feature extractor specification* in the CREATE INDEX and SELECT statements.

See "Creating Indexes" on page 4-8 for a description of the syntax for specifying a feature extractor function in the SQL CREATE INDEX statement.

See "Using the Resembles Function" on page 4-18 for a description of the syntax for specifying a feature extractor function in an SQL SELECT statement.

When Does Feature Extraction Occur?

During feature extraction, the database server invokes the feature extractor functions and the parameters that you specified when you created your index. Feature extraction occurs when you:

- create an **frnet** index on a table that has already been populated with images.
- insert or update a row in a table that contains a column that you indexed with an **frnet** index.
- perform a search with the **Resembles** function on a set of **frnet** indexed images.

When you create an **frnet** index, or insert or update a row in an **frnet** index, the database server invokes the feature extractor function specified in the EXTRACTOR parameter of the SQL CREATE INDEX statement. See "Creating an frnet Index" on page 4-9 for an example of the SQL CREATE INDEX syntax.

When you perform a search using the **Resembles** function, the database server invokes the specified feature extractor function to create a single feature vector from the search image. The feature extractor function tells the database server which index to use. The **Resembles** function is a strategy function that passes the image data, extracts the features, and conveys any other data needed by the **frnet** index to perform the search.

Creating Indexes

The Excalibur Image DataBlade module provides the **frnet** access method for creating feature-based indexes.

What Is the frnet Access Method?

The **frnet** access method allows you to create **frnet** indexes for any kind of digital information, including images, video, and sound.

You can create and search an **frnet** index on a column of any data type, provided the following two conditions are met:

- The data type of the input for the feature extractor function is the same as the data type of the column on which you want to create the **frnet** index.
- The feature extractor function generates an FrsFeatVect data type as output.

For example, suppose you store a collection of files in a column whose data type is a distinct BLOB TYPE. If your feature extractor function takes the BLOB TYPE as input and produces an FrsFeatVect data type as output, you can create an **frnet** index on the BLOB TYPE column and use it to search your database files.

The Excalibur Image DataBlade module uses the image feature vectors to create an **frnet** index. Each feature vector becomes a key in an **frnet** index. Figure 4-2 illustrates this process.

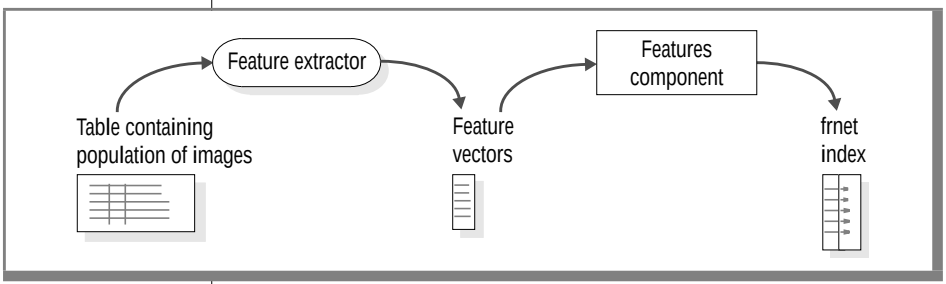


Figure 4-2
Process of Creating an frnet Index

After you create an **frnet** index, you can use it to search your images. The images to be searched are called *indexed images*. You can query the indexed images by passing a *search image* as a parameter of the **Resembles** function. See "Searching for Images" on page 4-16 for more information on the **Resembles** function.

Using the frnet Access Method

To use the **frnet** access method, create an **frnet** index on a database column of images. The **frnet** index is used to search for matching images.

For information about using the **frnet** access method to create an **frnet** index, see "Creating an frnet Index" on page 4-9. For information about querying a database table with an **frnet** index, see "Using the Resembles Function" on page 4-18.

Tip: *You can extend the* ***frnet*** *access method to perform content-based searches on other groups of database objects, such as acoustic recordings, using the* ***frnet*** *index.*

Creating an frnet Index

To create an **frnet** index, specify the **frnet** access method in the USING clause of the SQL CREATE INDEX statement. The syntax is shown in Figure 4-3.

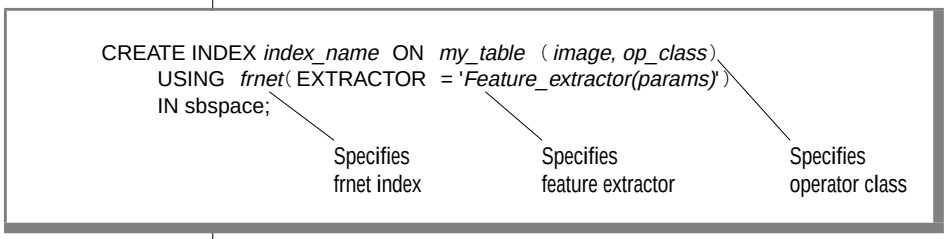


Figure 4-3 *SQL Statement Creating an frnet Index*

The entire expression in parentheses following **frnet** is also known as the *index parameter*.

You can create an **frnet** index on a column of any data type. However, a data type that has an **frnet** index associated with it must have one or more feature extractor functions and an operator class on the new data type, defining one or more strategy functions.

The following restrictions apply for all **frnet** indexes:

- When you create an **frnet** index, you cannot use the UNIQUE, DISTINCT, CLUSTER, ASC, DESC, or FILLFACTOR keywords.
- If you create your own feature extractor, it must return the FrsFeatVect data type.
- The **frnet** index must be a detached index where you specify a smart large object space (sbspace) in the IN clause of the SQL CREATE INDEX statement.

Tip: *All values for index creation options must be given as a VARCHAR data type.*

frnet Index Specification

A feature extractor specification consists of a feature extractor function name**,** the target image, and the list of feature extractor parameters. You can invoke a feature extractor function in an SQL CREATE INDEX statement.

index Parameter

The `index` parameter specifies the characteristics of an **frnet** index based on the searches you plan to perform. The `index` parameter is a comma-separated list of keyword value pairs. The keyword parameters for the **frnet** access method are:

- EXTRACTOR (required)
- HINT_MEMORY (optional)

The EXTRACTOR Keyword Parameter

The `EXTRACTOR` parameter is required for the **frnet** access method. It specifies the feature extractor to be used in index creation.

The value of the `EXTRACTOR` parameter is a quoted string. The string specifies the feature extractor name. It can also include parameters to be passed to the feature extractor. Different parameter settings result in different populations of feature vectors and, in turn, different **frnet** indexes. For more information about the feature extractor parameters that can be passed in the string, see "params Parameter" on page 4-12.

The HINT_MEMORY Keyword Parameter

The HINT_MEMORY parameter is optional. It specifies the number of kilobytes of memory a given index fragment occupies when the index is open for access.

The HINT_MEMORY parameter specifies the amount of memory allocated by the **frnet** index to cache index data for each index fragment. The default value is 2 megabytes per fragment. Each database user has an internal cache of index data. For example, 10 users querying the index simultaneously, each searching two index fragments, occupy 40 megabytes of memory. If memory is insufficient, some queries can fail until sufficient memory is freed. The more memory available to the **frnet** index, the faster the queries will run.

image Parameter

The `image` parameter identifies the image that serves as input to the feature extractor function. This image must be stored in an IfdImgDesc column. See "The IfdImgDesc Data Type" on page 2-4 for more information.

op_class Parameter

You must specify an operator class when you create an **frnet** index. For the Excalibur Image DataBlade module, the operator class for the **frnet** access method is IfdImgDesc_ops.

The **IfdImgDesc_ops** operator class must be specified in the SQL CREATE INDEX statement when an **frnet** index is created.

Tip: *You can use the* **IfdImgDesc_ops** operator class *to create new feature extractor functions.*

feature_extractor Parameter

The `feature_extractor` parameter is the feature extractor function to be invoked. The possible names are:

- IpdColorImageFE
- IpdGestaltImageFE
- IpdTextureImageFE

params Parameter

The `params` parameter is a comma-separated, keyword = value list. The `params` parameter sets the characteristics of the feature vectors. You can pass the following parameters to a feature extractor function:

- `RETINA_WIDTH`
- `RETINA_HEIGHT`
- `RETINA_TYPE`
- `RETINA_IMAGETYPE`

Tip: *Parameters are case insensitive.*

RETINA_WIDTH and RETINA_HEIGHT Parameters

The `RETINA_WIDTH` and `RETINA_HEIGHT` parameters allow you to specify the image size used for extracting feature vectors without altering the original image. Reducing the size of a set of images reduces the size of the corresponding feature vectors. This has the following advantages:

- Improved performance when feature vectors are created and retrieved because there are fewer bits to process
- Reduced disk space requirements for feature vectors

The optimal `RETINA_WIDTH` and `RETINA_HEIGHT` parameters vary for each feature extractor function, as follows:

- **IpdColorImageFE function.** The default setting is 32 pixels by 32 pixels. Feature vectors produced by the **IpdColorImageFE** function are the same size, regardless of the image size.
- **IpdGestaltImageFE function.** The default setting is 32 pixels by 32 pixels, for both unipole and dipole algorithms.
- **IpdTextureImageFE function.** The default setting is at least 200 by 200 pixels. The **IpdTextureImageFE** function compares neighborhood pixels to generate a feature vector. A very small image does not have enough pixels for a clear neighborhood bit comparison.

  A feature vector produced with the **IpdTextureImageFE** function for a given image is different from the feature vector produced by a resized version of the same image. Identical feature vectors result from a feature extractor function only if the images are the same size.

RETINA_TYPE Parameter

The `RETINA_TYPE` parameter determines the type of algorithm used by the **IpdGestaltImageFE** function. The `RETINA_TYPE` parameter has two possible values: `unipole` and `dipole`. The default value is `unipole`.

The unipole algorithm generates unique feature vectors for a wider range of images than the dipole algorithm, thus providing a generic method of creating feature vectors on an undefined range of images.

The following table compares the **IpdGestaltImageFE** function unipole and dipole algorithms.

Feature	Dipole Algorithm	Unipole Algorithm
Number of feature bits per byte of pixel data	1	2 (first bit = result from comparison and second bit = complement of first bit)
Comparison based on	2 image bytes	Values computed from resulting index
Processing speed	Faster than unipole	Slower because two comparisons performed
Feature vector content	Contains more 0s than 1s	Contains equal number of 0s and 1s
Feature vector size	4*((w*h*k+31)/32) bytes where w=image width, h=image height, and k=image type (1 for gray and 3 for RGB)	2 * size of (dipole feature vector)
Resulting index size	Size of (largest feature vector in indexed set) * (number of objects in indexed set) + sum (auxiliary data and file storage overhead)	2 * size of (dipole feature vector index)

(1 of 2)

Feature	Dipole Algorithm	Unipole Algorithm
Changes in features that are not reflected in a change in feature vector	Slight changes in rotation, translation, scale, brightness, and contrast Same changes in dissimilar images	Similar image brightness
Disadvantages	Similar images produce similar feature vectors Small number of unique pixels produce fewer bits (XrsSearch picks up 1 bits and not 0 bits, therefore bias created) Tolerant of changes in brightness	Results in larger feature vector indexes Requires longer time to create feature vectors
Summary	Overall brightness cannot be important Images should contain wide range of pixel values to generate unique feature vectors	Best suited for creating unique feature vectors on a wider range of images since features and contrast can be better characterized into unique feature vectors

(2 of 2)

Important: *The RETINA_TYPE parameter is available only for the* ***IpdGestaltImageFE*** *function.*

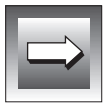
Warning: *Feature extraction using the unipole algorithm is slower and requires twice the amount of memory or disk space to store the resulting index.*

RETINA_IMAGETYPE Parameter

The `RETINA_IMAGETYPE` parameter defines the image type. There are two values available for image types: `GRAY` and `RGB`. The default setting is `GRAY`.

Feature vectors within a single index must be created from images of the same image type. Thus, images indexed must be converted to either GRAY or RGB type images. The **IpdGestaltImageFE** function converts the image to either GRAY or RGB.

If you are indexing primarily color images, choose the RGB image type so that the color table characteristics are included in the feature vector. If you are indexing primarily noncolor images (for example, BITMAP and GRAY), choose the default GRAY IMAGE type.



The conversion of images is done automatically by the **IpdGestaltImageFE** function. The image type is not permanently changed, but is only modified for that call of the function.

Important: *The RETINA_IMAGETYPE parameter is available only for the* ***IpdGestaltImageFE*** *function.*

Creating Multiple frnet Indexes

You can implement any number of **frnet** indexes on a given column, as long as they are unique. Different feature extractor specifications (feature extractor and parameters) result in different indexes. When an **frnet** index is searched, the feature extractor specification is needed to identify which **frnet** index to use in a search. Without the feature extractor specification, you cannot perform a search of the **frnet** index.

You cannot alter the feature extractor specification associated with an **frnet** index after you create the index. Instead, you must drop the index and create a new one with the desired feature extractor specification. When you drop an **frnet** index, the database server drops all index data.

Be sure to record the feature extractor specification that you use to create an **frnet** index so that when multiple **frnet** indexes exist on a column, you can specify the index to be searched.

Logging frnet Indexes

Always use the LOG option of the SQL CREATE TABLE statement for the smart large object spaces (sbspaces) where you plan to store your **frnet** indexes. See the *Informix Guide to SQL: Syntax* for the syntax of this statement.

Always run all SQL statements on tables containing an **frnet** index within a transaction to avoid inconsistencies in the event of a failure.

Creating Fragmented frnet Indexes

Fragmenting an **frnet** index can significantly improve performance. The Excalibur Image DataBlade module supports fragmented **frnet** indexes on columns and components of columns stored by value, but not those stored by reference.

Example

The following statement creates an **frnet** index using an image feature extractor for gestalt learning:

```
CREATE INDEX if_img on imgtable (image IfdImgDesc_ops)
USING frnet
(extractor=
'IpdGestaltImageFE(RETINA_WIDTH=40, RETINA_HEIGHT=40)')
FRAGMENT BY EXPRESSION
    img_id < 5 IN sbspace,
    img_id >= 5 AND img_id < 10 IN sbspace2,
    img_id >= 10 IN sbspace3;
```

Searching for Images

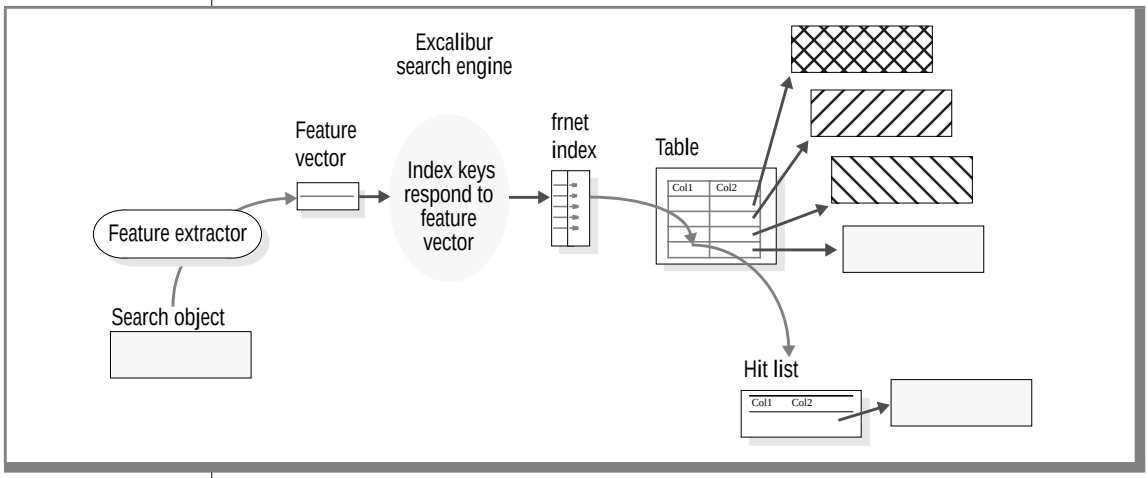
Before performing a search, the **frnet** access method calls a feature extractor function to extract the features of the search image and store them in a feature vector.

To perform the search, the query optimizer uses the feature extractor function specifications to determine if an **frnet** index is present. If the optimizer finds multiple **frnet** indexes on the sample column, it uses the feature extractor function specifications to determine which index to use.

After the **frnet** index has been identified, the **frnet** access method invokes the feature extractor function for the **frnet** index to extract the features of the search image into a feature vector.

Finally, the **frnet** access method performs a neural network search for similar patterns of feature vectors contained in the **frnet** index and returns the rows that contain similar images. Figure 4-4 illustrates the search process.

Figure 4-4
Image Search Process



The **Resembles** function is used to query the database. The **Resembles** strategy function is a part of the **IfdImgDesc_ops** operator class.

For more information about the **Resembles** function and the **frnet** access method, see "Creating Indexes" on page 4-8.

Using the Resembles Function

The **Resembles** function is never called directly: it is specified in the WHERE clause of an SQL SELECT statement. The following figure illustrates the SQL syntax for the **Resembles** function.

```
SELECT *, rank FROM table_name,
    WHERE Resembles
        (
        indexed_column_name,
        row(
            'feature_extractor(feature_extractor_params)'),
            search_object,
            search_options
        ),
        rank#REAL);
```

Figure 4-5 *SQL SELECT Statement with Resembles Function*

You can specify only one **Resembles** function in an SQL WHERE clause. The query fails if more than one is specified. You must use separate queries when you search for matching images using the color, texture, and shape feature extractor functions.

When the **Resembles** function is executed in a WHERE clause of an SQL SELECT statement, the following operation occurs:

1. The query optimizer uses the feature extractor function specifications you provide to determine if an **frnet** index is present to perform the query. If the optimizer finds multiple **frnet** indexes on the sample column, it uses the feature extractor function specifications to determine which index to use.
2. The **frnet** access method invokes the feature extractor function for the **frnet** index to extract the features of the search image into a feature vector.
3. The **frnet** access method uses a neural network search technique to search for similar feature vectors contained in the **frnet** index and returns the rows that contain similar images.

The Resembles Function Parameters

The **Resembles** function takes the following parameters:

- `indexed_column_name`
- `row`
- `rank`

indexed_column_name Parameter

The `indexed_column_name` parameter is the name of the column in the database table to be searched. This column must contain your **frnet** indexed objects. You cannot use a subquery to specify this value.

row Parameter

The `row` parameter is a row expression consisting of the following elements:

- *`feature_extractor`*. This element is required. It must be a string constant. The options of **feature_extractor** are described in "params Parameter" on page 4-12. The options must be the same as those used to create the index.
- *`search_object`*. This element is required. The search object must be an IfdImgDesc object stored in shared memory. The search object need not be in the table being searched.
- *`search_options`*. This element is optional. It specifies the options you want to pass to the feature extractor function or the **frnet** access method. These options are parsed by the **frnet** access method, which stores any index options, then appends the remaining options to the parameters list passed to the feature extractor function. The **frnet** access method recognizes one query option: MAX_MATCHES. This parameter sets the maximum number of matches allowed per index fragment. It can be an integer greater than 1.

rank Parameter

When you perform a search of an **frnet** index, some of the returned images are a closer match than others. The `rank` parameter indicates the degree of similarity between your search object and each of the rows returned. The returned rank of the search object varies between `0` and `1.0`, with `0` indicating no match and `1.0` indicating a perfect match. Values between `0` and `1.0` indicate approximate matches: the higher the value, the closer the match.

The `rank` parameter is a normalized score determined by adding the number of matches between the given feature vector and the search feature vector, and dividing by the maximum possible score. The maximum possible score is the score derived if the search feature vector was exactly the same as one already learned by the index.

To access ranking information, use a *statement local variable* (SLV). An SLV is limited to the statement in which you use it. The following example illustrates the **Resembles** function and the `rank` parameter syntax:

```
SELECT img_id, rank FROM imgtable
WHERE
Resembles(image, row
('IpdGestaltImageFE(RETINA_WIDTH=40, RETINA_HEIGHT=40)',
row(row(row('IFX_FILE', NULL::LLD_Lob, '/tmp/lenncp.tif'),
NULL::lvarchar), '', '', 0, 0, '')),
rank #REAL)
AND rank > 0.7;
```

In this example, the **Resembles** function compares the search image (**lenncp.tif**) with images stored in the indexed image column and returns those images whose similarity rank is greater than 0.7.

The statement local variable declaration is `rank #REAL` and the statement local variable expression is `rank > 0.7`. For more information on SLV, see the *Informix Guide to SQL: Syntax*.

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 GIFF 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 Image Foundation component.

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.
JPG	.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
EXIMG	N.A.	EXIMG is supported only if you are using the Excalibur Image DataBlade module. EXIMG is the in-memory form of the canonical data structure. By storing an image as a BLOB data type in EXIMG FORMAT, YOU AVOID THE REQUIRED conversion to EXIMG format when an image in a native file format is loaded into memory.
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)

THE EXIMG format is the internal format used for image processing by the Excalibur Image DataBlade module. It is a specification for the canonical data structure, a platform-independent binary representation of the structure of an image. The EXIMG format is advantageous only when you perform image processing or image indexing.

If you store your images in native format, for example TIFF or GIF, the Excalibur Image DataBlade module converts your images to the EXIMG format before it processes them. Because this conversion requires additional CPU cycles, you can improve the efficiency of image processing operations by maintaining your files in EXIMG format, thus making conversion unnecessary.

You can generate files that are in the EXIMG format using Excalibur Image DataBlade module functions. You can create and store an image in EXIMG format only if you store the image as a BLOB data type. You cannot read from a file in EXIMG format or write an image to a file in EXIMG format.

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

Feature Extractor Functions

This section contains reference manual pages for the feature extractor functions provided by the Excalibur Image DataBlade module.

Function	Description	Page Number
IpdColorImageFE	Extracts color-related features of an image and records them in a feature vector	page B-2
IpdGestaltImageFE	Extracts shape-related features of an image and records them in a feature vector	page B-5
IpdTextureImageFE	Extracts texture-related features of an image and records them in a feature vector	page B-8

Important: *You do not call these functions directly; rather, they are invoked in the SQL CREATE INDEX statement. See* *"Creating an frnet Index" on page 4-9* *for more information on the* ***frnet*** *index.*

IpdColorImageFE

Generates a feature vector based on the color content of an image stored in an IfdImgDesc data type.

Syntax

```
FrsFeatVect IpdColorImageFE(params)
    VARCHAR(128) params;
```

params Specifies the image parameters as a string of keywords and values, separated by commas (*RETINA_WIDTH, RETINA_HEIGHT*).

Usage

The **IpdColorImageFE** function is never called directly by a user or application. The **IpdColorImageFE** function is specified in one of the following SQL statements:

- The USING clause of a CREATE INDEX statement

  See "Creating an frnet Index" on page 4-9 for the syntax of the CREATE INDEX statement specifying a feature extractor.

- The WHERE clause of a SELECT statement

  See "Using the Resembles Function" on page 4-18 for the syntax of the SELECT statement specifying a feature extractor.

The **IpdColorImageFE** function extracts features and sets the bits of a feature vector according to the color content of the image. The location of color within the image is irrelevant.

All feature vectors generated by the **IpdColorImageFE** function have a constant size of 1404 bits, regardless of the type or dimension of an image.

The following table summarizes the information about the parameters (*params*) that you can pass to the **IpdColorImageFE** function.

Parameter Name	Parameter Type	Purpose	Rules	Default
RETINA_WIDTH	INTEGER	Alters the width of an image to the width (in pixels) that you specify.	Reduces the image to the specified width while maintaining the aspect ratio. If you also specify *RETINA_HEIGHT*, the parameter specifying the larger reduction in image size (the smaller image) is used.	32 pixels
RETINA_HEIGHT	INTEGER	Alters the height of an image to the height (in pixels) that you specify.	Reduces the image to the specified height while maintaining the aspect ratio. If you also specify *RETINA_WIDTH*, the parameter specifying the larger reduction in image size (the smaller image) is used.	32 pixels

Tip: *Parameters are case insensitive.*

Returns

On success, returns a feature vector of data type FrsFeatVect.

Example

In the SQL CREATE INDEX statement, specify the **IpdColorImageFE** function in the USING clause, as shown in the following example:

```
CREATE INDEX ic_image on imgtable (image IfdImgDesc_ops)
USING frnet(extractor=
    'IpdColorImageFE(RETINA_WIDTH=32,  RETINA_HEIGHT=32)')
in sbspace;
```

In the SQL SELECT statement, specify the **IpdColorImageFE** function with the **Resembles** function in the WHERE clause, as shown in the following example:

```
SELECT img_id, rank FROM imgtable
WHERE
Resembles(image, row
('IpdColorImageFE(RETINA_WIDTH=32,  RETINA_HEIGHT=32)',
row(row(row
    ('IFX_FILE',
     NULL::LLD_Lob,
    '/tmp/lenncp.tif'),
NULL::lvarchar), '', '', 0, 0, '')),
rank #REAL);
```

IpdGestaltImageFE

Generates a feature vector based on the shape or outline of an image stored in an IfdImgDesc data type.

Syntax

```
FrsFeatVect IpdGestaltImageFE(params)
    VARCHAR(128) params;
```

params Specifies the image parameters as a string of keywords and values, separated by commas (*RETINA_WIDTH, RETINA_HEIGHT, RETINA_TYPE, RETINA_IMAGETYPE).*

Usage

The **IpdGestaltImageFE** function is never called directly by a user or application. It is specified in one of the following SQL statements:

- The USING clause of a CREATE INDEX statement

  See "Creating an frnet Index" on page 4-9 for the syntax of the CREATE INDEX statement specifying a feature extractor.

- The WHERE clause of a SELECT statement

  See "Using the Resembles Function" on page 4-18 for the syntax of the SELECT statement specifying a feature extractor.

The **IpdGestaltImageFE** function extracts features and sets the bits of a feature vector according to the shapes or outlines of an image. The **IpdGestaltImageFE** function provides a form of feature extraction that works most effectively with search images that—on the whole—resemble those in the **frnet** index.

The following table summarizes the information about the parameters (*params*) that you can pass to the **IpdGestaltImageFE** function.

Parameter Name	Parameter Type	Purpose	Rules	Default
RETINA_WIDTH	INTEGER	Alters the width of an image to the width (in pixels) that you specify.	Reduces the image to the specified width while maintaining the aspect ratio. If you also specify *RETINA_HEIGHT*, the parameter specifying the larger reduction in image size (the smaller image) is used.	32 pixels
RETINA_HEIGHT	INTEGER	Alters the height of an image to the height (in pixels) that you specify.	Reduces the image to the specified height while maintaining the aspect ratio. If you also specify *RETINA_WIDTH*, the parameter specifying the larger reduction in image size (the smaller image) is used.	32 pixels
RETINA_TYPE	CHAR(4)	Selects an extraction algorithm.	Valid values are: - UNIPOLE - DIPOLE See "RETINA_TYPE Parameter" on page 4-13 for details.	UNIPOLE
RETINA_IMAGETYPE	CHAR(4)	Selects an image type.	Valid values are: - GRAY - RGB	GRAY

Tip: *Parameters are case insensitive.*

Returns

On success, returns a feature vector of data type FrsFeatVect.

Example

In the SQL CREATE INDEX statement, specify the **IpdGestaltImageFE** function in the USING clause, as shown in the following example:

```
CREATE INDEX ig_img on imgtable (image IfdImgDesc_ops)
USING frnet(extractor=
    'IpdGestaltImageFE(RETINA_WIDTH=40, RETINA_HEIGHT=40)')
in sbspace;
```

In the SQL SELECT statement, specify the **IpdGestaltImageFE** function with the **Resembles** function in the WHERE clause, as shown in the following example:

```
SELECT img_id, rank FROM imgtable
WHERE
Resembles(image, row
('IpdGestaltImageFE(RETINA_WIDTH=40, RETINA_HEIGHT=40)',
    row(row(row
        ('IFX_FILE',
         NULL::LLD_Lob,
         '/tmp/lenncp.tif'),
    NULL::lvarchar), '', '', 0, 0, '')),
    rank #REAL);
```

IpdTextureImageFE

Generates a feature vector based on the texture of an image.

Syntax

```
FrsFeatVect TextureImageFE(params)
    VARCHAR(128) params;
```

params Specifies the image parameters as a string of keywords and values, separated by commas (*RETINA_WIDTH*, *RETINA_HEIGHT*).

Usage

The **IpdTextureImageFE** is never called directly by a user or application. It is specified in one of the following SQL statements:

- The USING clause of a CREATE INDEX statement

  See "Creating an frnet Index" on page 4-9 for the syntax of the CREATE INDEX statement specifying a feature extractor.

- The WHERE clause of a SELECT statement

  See "Using the Resembles Function" on page 4-18 for the syntax of the SELECT statement specifying a feature extractor.

The **IpdTextureImageFE** function extracts features and sets the bits of a feature vector according to the texture of an image. The location of textures within the image is irrelevant.

The **IpdTextureImageFE** function generates a feature vector of 512 bits for every image that it processes, regardless of the type or dimension of the image.

The following table summarizes the information about the parameters (*params*) that you can pass to the **IpdTextureImageFE** function.

Parameter Name	Parameter type	Purpose	Rules	Default
RETINA_WIDTH	INTEGER	Alters the width of an image to the width (in pixels) that you specify.	Reduces the image to the specified width while maintaining the aspect ratio. If you also specify *RETINA_HEIGHT*, the parameter specifying the larger reduction in image size (the smaller image) is used.	200 pixels
RETINA_HEIGHT	INTEGER	Alters the height of an image to the height (in pixels) that you specify.	Reduces the image to the specified width while maintaining the aspect ratio. If you also specify *RETINA_WIDTH*, the parameter specifying the larger reduction in image size (the smaller image) is used.	200 pixels

Tip: *Parameters are case insensitive.*

Returns

On success, returns a feature vector of data type FrsFeatVect.

Example

In the SQL CREATE INDEX statement, specify the **IpdTextureImageFE** function in the USING clause, as shown in the following example:

```
CREATE INDEX it_image on imgtable (image IfdImgDesc_ops)
USING frnet(extractor=
'IpdTextureImageFE(RETINA_WIDTH=200, RETINA_HEIGHT=200)')
in sbspace;
```

In the SQL SELECT statement, specify the **IpdTextureImageFE** function with the **Resembles** function in the WHERE clause, as shown in the following example:

```
SELECT img_id, rank FROM imgtable
WHERE
Resembles(image, row(
'IpdTextureImageFE(RETINA_WIDTH=200, RETINA_HEIGHT=200)',
row(row(row('IFX_FILE', NULL::LLD_Lob, '/tmp/lenncp.tif'),
NULL::lvarchar), '', '', 0, 0, '')),
rank #REAL);
```

Appendix

C

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 Image Foundation component 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 following is the type definition for the IfdImage data structure:

```
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 Image Foundation component. The Image Foundation component 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 Image Foundation component. By default, all Image Foundation component 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 following is the type definition of the IfdIpsImg data structure:

```
/* in-memory image structure */
typedef struct
        {
        mi_integer      width;
        mi_integer      height;
        mi_pointer      data;
        uint4           compression;
        uint4           photometric;
        mi_double_precision xdpi;
        mi_double_precision ydpi;
        uint4           samples_per_pixel;
        uint4           bits_per_sample[3];
        uint4           bit_order;
        uint4           unitsz;
        uint4           nunits;
        uint4           additives;
        uint1           *ctab;
        mi_integer      nctab;
        } IFDimage;

#define IfdIpsImg IFDimage
```

The following table describes the fields of the IfdIpsImg structure.

width	*x*-dimension of the image, in pixels.
height	*y*-dimension of the image, in pixels.
data	Pointer to pixel data. The data is organized in units of unsigned bytes (8-bit integer). If the image is not compressed, each row is padded to the nearest byte boundary, and the number of bytes per row in an image is consistent. If the image data is compressed, IfdIpsImg data is a pointer to the compressed data. In this case, the number of bytes per row is unknown and most likely will vary from row to row. It can be as little as one bit (G42 compression for a row of white pixels identical to the previous row).

compression	Specification of how to interpret IfdIpsImg.data. When compression is equal to 0, the data is not compressed. Refer to the **$INFORMIXDIR/incl/ifd.h** file for a list of supported compression codes.
photometric	For BINARY images, this member must be 0 when compression is 0, otherwise it can be either 1 or 0. If the value is 1/0, a pixel code of 0 represents black / white. This member is set to 1 for image types CTAB and RGB and 3 for HSV. Example: a G42 compressed image has a photometric value equal to 1, the image is uncompressed, 0 represents black, and 1 represents white. Uncompressed BINARY images always use 0 to encode white and 1 to encode black. Therefore, a 0 emitted from the uncompress routine is converted to 1 before being stored in data, and photometric is set to 0. Uncompressed BINARY images always use 0 to reduce the number of cases binary image-processing algorithms handle.
xdpi, ydpi	The *x* and *y* resolution of the image, measured in dots per inch.
samples_per_pixel	Samples (or components) per pixel code: BINARY, GRAY, and CTAB—1 RGB and HSV—3 This value is the number of valid entries in the *bits_per_sample[]* member.
bits_per_sample[3]	The number of bits per sample: BINARY—1,0,0 GRAY and CTAB—8,0,0 RGB and HSV—8,8,8

bit_order	The order of storage of a pixel code in a storage unit. The *bit_order* value is 0 for compressed BINARY images; otherwise, it must always be 1. Uncompressed BINARY images always use 1 to reduce the number of cases BINARY image processing algorithms handle. The pixel code of a BINARY image is one bit and the byte has space for 8 bits, thus it is possible to store the first pixel code at the least significant position (0 x 01, 0) or most significant position (0 x 80, 1) of the byte.
unitsz	This must be 1 for all image types.
nunits	Size in bytes of the buffer pointed to by **IfdIpsImg.data**. This member must always be set, even though the value for uncompressed images can be calculated from the image type, width, and height.
additives	Returns the format code of the image and is required by the **WriteFromMem** and the **WriteLocFromMem** functions. Refer to the **$INFORMIXDIR/incl/ifd.h** file for a list of supported compression codes.
ctab	Pointer to a buffer containing a color table. If **IpsImg.ctab** is NULL, the image has no color table; otherwise, the image type is CTAB. The maximum number of entries in the table is 256. Each color table entry uses three unsigned bytes: First byte—red component Second byte—green component Third byte—blue component The length of the color table in bytes is 3 times its number of entries. For example, ctab[17] contains the blue component of the color at color table entry 5.
nctab	The number of entries in *ctab*. This value is 0 if *ctab* is NULL.

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

Image Type	Canonical Format Representation
Binary	1-bit representation
Gray	8-bit representation
CTAB	8-bit representation
RGB	24-bit representation
HSV	24-bit representation (supported in EXIMG format)

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

Image Type	Bits Per Pixel Code	Bytes Per Row	Pixel Code Arrangement	Photometric
BINARY	1	(width+7)/8	Order is from most to least significant bit within a byte.	0 is white, 1 is black. (IFD_ZERO_WHITE)
GRAY	8	width	The entire byte is the pixel code.	0 is black, 255 is white. (IFD_ZERO_BLACK)
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_ZERO_BLACK)
RGB	8,8,8	3*width	One byte each for red, green, and blue, in that order.	The color read in data. (IFD_ZERO_BLACK)
HSV	8,8,8	3*width	One byte each for hue, saturation, and value, in that order.	The color read in data. (IFD_HSV_DATA)

The IfdImagePtr Data Type

The IfdImagePtr data type is a distinct pointer used by the Image Foundation component to point to the IfdImage structure. This type of pointer is required by the following functions supplied by the Image Foundation component:

- **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 following is the type definition for the IfdImagePtr data type:

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

Functions

This section contains reference pages for the functions supplied by the Image Foundation component for DataBlade module developers. These functions operate on the C structures described in "Data Structures and Types" on page C-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 C-9

(1 of 2)

ReadToMem	Reads an image from the source location into an IfdImage structure in shared memory	page C-10
WriteFromMem	Writes an image in shared memory to a destination IfdImgDesc	page C-12
WriteLocFromMem	Writes an image in shared memory to a destination IfdLocator	page C-14

(2 of 2)

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, uncompress,
translate_function)
    IfdImgDesc image;
    VARCHAR(25) imgformat;
    INTEGER frame_no;
    mi_boolean uncompress;
    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).
uncompress	Specifies whether the image should be uncompressed (if saved in compressed format) using the compression algorithm with which the image was saved.
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 following values are valid for the *uncompress* parameter:

- `T` (true). If the image is compressed, uncompress it.
- `F` (false). Return an image without uncompressing it. If an image is saved compressed, it is returned compressed in-memory.

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

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.
compression	Specifies the compression scheme. Values can be `NULL`, `AUTO[:factor]`, or, for JFIF format, `JPEG[:factor]`.

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 Image Foundation component or other image-based user-defined routines 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.

The following table summarizes the possible values for the *compression* parameter.

Value	Definition
NULL	Do not compress the image before writing it to the IfdLocator structure.
AUTO[:factor]	Use the compression algorithm that uncompressed the image in the **ReadToMem** function (as specified in the IfdMemImg compression field) for JFIF file formats.
JPEG[:factor]	Represents the size and quality trade-off value (the default 75 is currently used in the JPEG compression algorithm). See "ImgCompression" on page 3-9 for the available compression schemes.

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

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 image.
compression	Specifies the compression scheme. Values can be `NULL`, `AUTO[:factor]`, or, for JFIF format, `JPEG[:factor]`.

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 Image Foundation component 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.

The following table summarizes the possible values for the *compression* parameter.

Value	Definition
NULL	Do not compress the image before writing it to the IfdLocator structure.
AUTO[:factor]	Use the compression algorithm that uncompressed the image in the **ReadToMem** function (as specified in the IfdMemImg compression field) for JFIF file formats.
JPEG[:factor]	Represents the size and quality trade-off value (the default 75 is currently used in the JPEG compression algorithm). See "ImgCompression" on page 3-9 for the available compression schemes.

Returns

On success, returns an IfdLocator structure.

Appendix

D

C Code Examples

This appendix contains the C code examples for the functions supplied by the Image Foundation component for DataBlade module developers. See Appendix C for the function reference pages.

Function	Description	Page Number
FreeImagePtr	Frees an IfdImagePtr pointer	page C-9
ReadToMem	Reads an image from the source location into an IfdImage structure in shared memory	page C-10
WriteFromMem	Writes an image in shared memory to a destination IfdImgDesc	page C-12
WriteLocFromMem	Writes an image in shared memory to a destination IfdLocator	page C-14

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

Image Foundation Component C Code Examples

The following C code contains examples of the user-defined routines for the Image Foundation component of 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 Image Foundation 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 Image Foundations Datablade'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 Image Foundations Datablade'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 Image Foundations Datablade'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

dipole One of two feature extraction algorithms used by the gestalt *feature extractor function* to extract *feature vectors* based on the shape of objects contained in the image. The choice of *unipole* or dipole algorithms is specified with the *RETINA_TYPE* parameter. The unipole algorithm is the default. The dipole algorithm is better suited to a population of images that contains a wide range of pixel values, where overall brightness is not important. The dipole algorithm requires less processing time and creates smaller feature vectors and indexes than the unipole algorithm. The dipole algorithm produces similar feature vectors for similar images.

feature extractor function A function that decodes the important attributes of an object into *feature vectors*. The Excalibur Image DataBlade module provides color, shape (gestalt), and texture *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 extractor specification A specification that consists of a *feature extractor function* name, the data type of the image or other object whose features are being extracted, and a list of *feature extractor parameters*.

feature vector An array of bits representing the presence or absence of specific features. Feature vectors are created by a *feature extractor function* and indexed by the *frnet access method*.

frnet access method A secondary access method provided by the Excalibur Image DataBlade module. The *frnet access method* uses neural network searches to match images.

frnet index An index created on feature vectors of images or other objects by the *frnet access method*.

gestalt An image or pattern so unified that its properties cannot be derived from its parts. The **IpdGestaltImageFE** *feature extractor function* derives its name from the term *gestalt* because it works most effectively with a search image that, on the whole, resembles the *indexed images*.

indexed image The images in a database table indexed by the *frnet access method*.

operator class The set of functions (operators) that Informix Dynamic Server with Universal Data Option associates with an access method. The operator class for the Excalibur Image DataBlade module is defined on the IfdImgDesc data type and used by the **frnet** access method.

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

signature The information the database server uses to identify a routine. The signature of a routine includes the type of routine, the routine name, the number of parameters, the data types of the parameters, and the order of the parameters. The database server uses the signature of a *feature extractor function* (specified as a parameter of the **Resembles** operator function) to identify a particular *frnet index*.

statement local variable (SLV) A variable whose scope is limited to the statement in which it is used. The *RANK* parameter of the **Resembles** operator function is an SLV.

unipole One of two feature extraction algorithms used by the **IpdGestaltImageFE** *feature extractor function* to extract *feature vectors* based on the shape of objects contained in the image. The choice of unipole or *dipole* algorithms is specified with the *retina_type* parameter. The unipole algorithm is the default because it creates unique feature vectors for an undefined range of images. However, the unipole algorithm requires more processing time and creates larger feature vectors and indexes than the dipole algorithm.

Index

G

H

I

L

R

S

U

W