DB2 Universal Database Image, Audio,
and Video Extenders

Administration and Programming

Version 5 Release 2

SC26-9107-01

DB2 Universal Database Image, Audio,
and Video Extenders

Administration and Programming

Version 5 Release 2

SC26-9107-01

Note

Before using this information and the product it supports, please read the general information under “Appendix D. Notices” on page 569.

This edition replaces and makes obsolete SC26-9107-00. The technical changes for this edition are summarized under “Summary of Changes”, and are indicated by a vertical bar to the left of the change.

Contents

Figures

Tables

About this book

This book describes how to use DB2 extenders to prepare and maintain a DB2 database for image, audio, or video data. It also describes how you can use user-defined functions (UDFs) and application programming interfaces (APIs) provided by DB2 extenders to access and manipulate these types of data. By incorporating UDFs in your program's SQL statements, and incorporating APIs, you can access nontraditional data, such as images and video clips, and traditional numeric data and character data.

References in this book to ″DB2″ refer to DB2 Version 5.2 or higher.

Who should use this book

This book is intended for DB2 database administrators who are familiar with DB2 administration concepts, tools, and techniques.

This book is also intended for DB2 application programmers who are familiar with SQL and with one or more programming languages that can be used for DB2 application programs.

This book is for people who will work with the DB2 Image, Audio, and Video Extenders. People who work with the Text Extender should see *DB2 Text Extender Administration and Programming.*

How to use this book

This book is structured as follows:

"Part 1. Introduction".

This part gives an overview of the DB2 extenders . Read this part if you are new to administering or programming with the DB2 extenders .

"Part 2. Administering Image, Audio, and Video Data".

This part describes how to prepare and maintain a DB2 database for image, audio, and video data. Read this part if you need to administer a DB2 database that contains image, audio, or video data.

"Part 3. Programming for Image, Audio, or Video Data".

This part describes how to use the DB2 extender UDFs and APIs to request operations on image, audio, or video data. Read this part if you need to access and manipulate image, audio, or video data in a DB2 application program.

"Part 4. Reference"

This part presents reference information for DB2 extender UDFs, APIs, administrative commands, and diagnostic information such as messages and codes. Read this part if you are familiar with DB2 extender concepts and tasks, but need information about a specific DB2 extender UDF, API, command, message, or code.

"Appendixes"

The appendixes describe:

- How to set environment variables that are used by the DB2 extenders to find files and to identify display or player programs for image, audio, and video objects
- How to install and use sample programs and media files that are provided with the extenders
- How to migrate to DB2 extenders V5 from a previous version

Platform-specific information

DB2 Extenders can be used in conjunction with the single-partition database environment of DB2 Universal Database, or with the multi-partition database environment of DB2 Universal Database Extended Enterprise Edition.

This book contains information on using DB2 extenders in either environment. Information that pertains only to using the extenders in the multipartition environment of DB2 Universal Database Extended Enterprise Edition is marked ″**EEE Only.**″ Information that pertains only to using the extenders in the single partition environment of DB2 Universal Database is marked ″**Non-EEE Only.**″ Information that is not marked as pertaining to a specific environment applies to both environments.

Highlighting conventions

This book uses the following conventions:

Bold Bold text is used to indicate a definition of a new term.

Italics Italics indicate variable parameters that are to be replaced with a value, or it emphasizes words that are used in text.

UPPERCASE

Uppercase letters indicate:

- Data types
- Directory names
- Field names

- API calls
- Commands
- Keywords
- Variable names

Example

Example text indicates a system message or value you type. Example text is also used for coding examples.

How to read the syntax diagrams

Throughout this book, command, and SQL syntax are described using syntax diagrams. Read the syntax diagrams as follows:

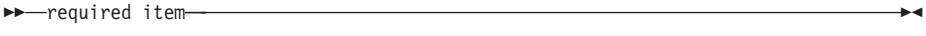
- Read the syntax diagrams from left to right and top to bottom, following the path of the line.

  The ▶▶— symbol indicates the beginning of a statement.

  The —▶ symbol indicates that the statement syntax is continued on the next line.

  The ▶— symbol indicates that a statement is continued from the previous line.

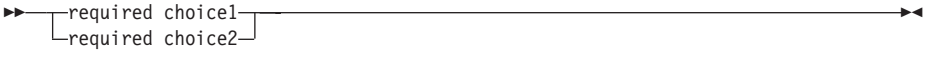
  The —▶◀ symbol indicates the end of a statement.
- Required items appear on the horizontal line (the main path).



- Optional items appear below the main path.

- If you can choose from two or more items, they appear in a stack.

  If you *must* choose one of the items, one item of the stack appears on the main path.



  If choosing none of the items is an option, the entire stack appears below the main path.

A repeat arrow above a stack indicates that you can make more than one choice from the stacked items.

- Keywords appear in uppercase (for example, `/DB2IMAGE:`). They must be spelled exactly as shown. Variables appear in lowercase (for example, `srcpath`). They represent user-supplied names or values in the syntax.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Related information

DB2 Universal Database Version 5

DB2 Universal Database Version 5 Quick Beginnings , S10J-8147 (OS/2), S10J-8149 (Windows), S10J-8148 (UNIX). These books describe how to plan for and install DB2 on the appropriate platform.

DB2 Universal Database Version 5 Administration Guide , S10J-8157. This book describes how to design, create, and maintain a DB2 database.

DB2 Universal Database Version 5 Embedded SQL Programming Guide , S10J-8158. This book describes the application development process, and how to code, compile, and run programs that use embedded SQL and APIs to access a DB2 database.

DB2 Universal Database Version 5 Call Level Interface Guide and Reference , S10J-8159. This book describes how to use DB2 CLI in applications to access DB2 servers.

DB2 Universal Database Version 5 Command Reference , S10J-8166. This book gives reference information about commands that are used to perform DB2 administrative tasks.

DB2 Universal Database Version 5 Messages Reference , S10J-8168. This book lists DB2 messages that identify both errors or problems, and recovery actions.

IBM DB2 Universal Database Extended Enterprise Edition

IBM DB2 Universal Database Extended Enterprise Edition for Windows NT Quick Beginnings Version 5 , SO9L-6713-00. This book describes how to

install and use the basic functions of DB2 Universal Database Extended Enterprise Edition in a Windows NT operating environment.

IBM DB2 Universal Database Extended Enterprise Edition for UNIX Quick Beginnings Version 5 , S99H-8314-00. This book describes how to install and use the basic functions of DB2 Universal Database Extended Enterprise Edition in a UNIX operating environment.

DB2 Text Extender

DB2 Universal Database Text Extender Administration and Programming , SC26-9108. This book describes how to administer a DB2 database for text data. It also describes how to use application programming interfaces that are provided by the DB2 Text Extender to access and manipulate text data.

World Wide Web

DB2 extenders web site. This web site contains information about the DB2 extenders as well as technologies that are pertinent to the extenders. The URL of the DB2 extenders home page is: http://www.software.ibm.com/data/db2/extenders.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 extenders documentation. You can use any of the following methods to provide comments:

- Send your comments from the Web. Visit the Web site at:

  `http://www.software.ibm.com/data/db2/extenders`

  The Web site has a feedback page that you can use to enter and send comments.
- Send your comments by e-mail to comments@vnet.ibm.com. Be sure to include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a chapter and section title, a table number, a page number, or a help topic title).
- Mail comments to:

  IBM Corporation,
  Department HHX/H3
  P.O. Box 49023
  San Jose, CA 95161-9023
  U.S.A.
- Fax comments to 800-426-7773 (in the United States or Canada).

- Give comment to an IBM representative.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Summary of Changes

The following major changes have been made to the document for this edition:

- **DB2 Universal Database Extended Enterprise Edition.** DB2 extenders operate with and exploit the partitioned database support of DB2 Universal Database Extended Enterprise Edition in Windows NT, Solaris Operating Environment, and AIX platforms. This means that extender data can be distributed across multiple machines, and queries can be run in parallel, thereby speeding up transaction processing. Changes have been made throughout this document that cover using DB2 extenders in a partitioned database environment. Information that is unique to using DB2 extenders in a partitioned database environment is marked **EEE only**.

Part 1. Introduction

Chapter 1. Overview

DB2 (DB2) Universal Database (UDB) Version 5 is a powerful, object-relational database manager. It stores and protects traditional numeric and character data, as well as large, complex objects (LOBs). DB2 extenders help you exploit DB2 Version 5's object-relational features. The extenders define distinct data types and special functions for image, audio, video, and text objects. By doing this, the extenders save you the time and effort of defining these data types and functions in your applications. The data types and functions are available through SQL. Because of that, the extenders give your applications a single point of access to any or all of these types of data, along with traditional numeric and character data. In addition, the extenders give your applications new ways to search for information. For example, your applications can search for images by their visual characteristics, using visual examples of color or texture.

Exploiting DB2 Version 5

The DB2 extenders exploit the object-oriented features of DB2 Version 5. In particular, with DB2 Version 5 you can:

- Store LOBs of up to 2 gigabytes in a DB2 database.
- Define distinct data types for these large, complex objects. You use these user-defined types (UDTs) to identify the type of data that is represented by an object, for example, an image or an audio.
- Define specific functions that can be requested on a user-defined type of data. For example, you can define a function to count the number of colors in an image or to get the sampling rate of an audio. You request these user-defined functions (UDFs) in an SQL statement in the same way as other SQL functions.

The DB2 extenders create UDTs and UDFs for image, audio, video, and text objects. The UDTs and UDFs can be important aids in helping you:

- Develop applications. Because the extenders define the data types and functions, you do not have to define them in your applications.
- Ensure consistency. The same set of extender UDTs and UDFs are available to all of your applications. This offers a ready-made level of consistency that might otherwise be difficult to achieve across applications that handle large objects.
- Create powerful queries. Because the UDFs are invoked in the same way as other SQL functions, your applications can include multi-data-type queries. One SQL statement can access image, audio, video, and text objects,

together with traditional numeric and character data. You can specify UDFs and UDTs in embedded SQL statements as well as in DB2 Call Level Interface (DB2 CLI) calls.

And because the objects that the extenders process can be stored in a DB2 database, the same security, integrity, and recovery protections are in place for those objects as for traditional data types stored in the database.

In addition, the DB2 extenders exploit the partitioned database environment of DB2 Extended Enterprise Edition. Partitioning allows applications to use a database that is too large for a single computer, and allows SQL operations to perform in parallel, thereby speeding up SQL queries or utilities.

Powerful new ways to search for information

The DB2 extenders give your applications a lot of flexibility in searching for information. Your applications can search for objects that are associated with traditional types of data that are stored in a database. For example, they can search for an audio clip by its description or by the date it was recorded. Your applications can also search for objects by their inherent characteristics, such as the playing time of a video clip. The extenders automatically determine and store these characteristics for use in searches.

Your applications can even search for images by content. Imagine an application that uses visual examples to search for images. With such an application, users could select an example image and have the application find other images that have colors or textures similar to those in the example. With DB2 extenders' Query by Image Content (QBIC) capability, you can create applications that search for images in this visual way.

The DB2 extenders

The DB2 extenders comprise a separate Image Extender, Audio Extender, Video Extender, and Text Extender.

This book covers the Image, Audio, and Video Extenders. All further references to "extenders" or "DB2 extenders" in this book refer to the Image, Audio, and Video Extenders, unless otherwise noted. For information about the Text Extender, see *Text Extender Administration and Programming*.

Using the extenders

You can invoke the extender UDFs in a DB2 application program, or you can invoke them interactively using the DB2 command-line processor.

The extenders also provide the following application programming interfaces (APIs):

- Administrative APIs to prepare and maintain a database for image, audio, and video data
- Display and play APIs to display images and play video and audio clips
- QBIC APIs to prepare images for, and request searches by content. (A content search can also be requested through UDFs.)
- Video shot detection APIs to identify sequences of frames that are based on scene changes in a video

Unlike UDFs, which are requested in SQL statements, the administrative APIs, display and play APIs, QBIC APIs, and video shot detection APIs are requested through API calls.

The DB2 extenders also provide a command-line processor that you use to issue administrative commands. To differentiate the command-line processor provided by the extenders from the command-line processor provided by DB2, we'll refer to the former as the "db2ext command-line processor" and the latter as the "DB2 command-line processor".

Examples

An advertising agency maintains a DB2 database of its advertisements. In the past, the agency stored numeric and character data about each ad campaign, such as the name of the client and the date that an advertisement was completed. With the installation of DB2 Version 5 and the DB2 extenders, the agency now also stores the content of the ads in the database. This includes images of print ads, videos of television ads, and recordings of radio ads. As Figure 1 on page 6 shows, all of the related advertising information is in one database table that is named ADS.

Examples

Figure 1. A multimedia database table. The table contains image, audio, and video data as well as traditional data types. A video, audio, and image are shown.

Example 1: Retrieving a video by its characteristics

An account manager in the advertising agency needs to see the video ads created for the IBM account in 1997, but only ads whose duration is 30 seconds or less.

Figure 2 on page 7 shows a query that accesses the videos. Notice that the Video Extender UDFs named `Filename` and `Duration` in the query.

```
SELECT Filename(Ads_video)
FROM ADS
WHERE Client='IBM' AND
Ship_date≥'01/01/1997' AND
Duration(Ads_video) ≤30
```

Figure 2. A query that accesses videos

The query returns the file names of the desired videos. The account manager can then start his favorite video player and play the content of each video file.

Figure 2 is an example of a query that the account manager can issue interactively. More typically, the account manager would use an application program to find and play videos. For example, Figure 3 on page 8 shows some key elements of such an application coded in C. The application retrieves the video file names in a DB2 host variable named `hvVid_fname`. Also notice that the application uses a play API, named `DBvPlay`, to play the videos.

Examples

```
#include <dmbvideo.h>

int count = 0;

EXEC SQL BEGIN DECLARE SECTION;
char hvClient[30];                      /*client name*/
char hvCampaign[30];                    /*campaign name*/
char hvSdate[8];                        /*ship date*/
char hvVid_fname [251]                  /*video file name*/
EXEC SQL END DECLARE SECTION;


EXEC SQL DECLARE c1 CURSOR FOR
     SELECT CLIENT, CAMPAIGN, SHIP_DATE, FILENAME(ADS_VIDEO)
       FROM ADS
       WHERE CLIENT='IBM' AND
         SHIP_DATE≥'01/01/1997' AND
         DURATION(ADS_VIDEO)≤30
FOR FETCH ONLY;

EXEC SQL OPEN c1;
for (;;){
        EXEC SQL FETCH c1 INTO :hvClient, :hvCampaign,
                         :hvSdate, :hvVid_fname;

    if (SQLCODE != 0)
              break;

    printf("\nRecord %d:\n", ++count);
    printf("Client = '%s'\n", hvClient);
    printf("Campaign = '%s'\n", hvCampaign);
    printf("Sdate = '%s'\n", hvSdate);

    rc=DBvPlay(NULL,MMDB_PLAY_FILE,hvVid_fname,MMDB_PLAY_WAIT);
}
EXEC SQL CLOSE c1;
```

Figure 3. An application that accesses and plays videos

Example 2: Searching for images by content

A graphic illustrator in the advertising agency is developing a new print ad for a client. The illustrator wants to use a particular shade of blue in the background of the ad, and wants to see if the color has been used before in printed advertisements created by the agency. To do that, the graphic illustrator runs an application that searches for images by content. The images are stored in a database table (see Figure 1 on page 6). The application asks the user to supply a visual example, that is, an image that demonstrates the color of interest. The application then analyzes the color in the example and finds images whose color best matches the example.

Figure 4 shows a visual example and the retrieved images that most closely match its color.

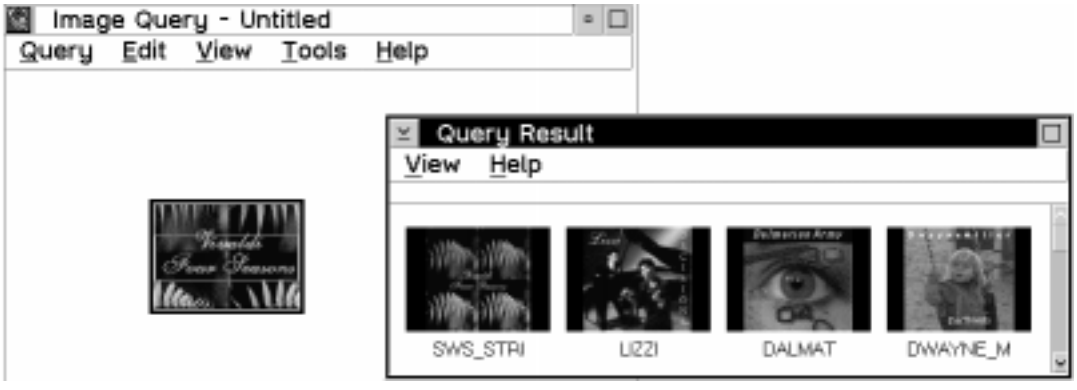


Figure 4. Searching for images by content. A visual example is used to search for images by average color.

Figure 5 on page 10 shows some key elements of the application. Notice that the application uses a QBIC API named `QbQueryCreate` to create a QBIC query, `QbQueryAddFeature` and `QbQuerySetFeatureData` to add the color selection to the query, `QbQuerySearch` to issue the query, and `QbQueryDelete` to delete the query. The application also uses a graphical API, named `DBiBrowse`, to display the retrieved images.

Examples

```
#include <dmbqbqpi.h>

#define MaxQueryReturns  10

static SQLHENV henv;
static SQLHDBC hdbc;
static SQLHSTMT hstmt;
static SQLRETURN rc;

void main(int argc, char* argv[])
{
 char           line[4000];
 char*          handles[MaxQueryReturns];
 QbQueryHandle  qHandle=0;
 QbResult       results[MaxQueryReturns];
 SQLINTEGER     count;
 SQLINTEGER     resultType=qbiArray;

 SQlAllocEnv(&henv);
 SQLAllocConnect(henv, &hdbc);
 rc = SQLConnect(hdbc, (SQLCHAR*)"qtest", SQL_NTS,
              (SQLCHAR*)"", SQL_NTS, (SQLCHAR*)"", SQL_NTS);

 if (argc !=2) {
 printf("usage: query colorname\n");
          exit(1);
 }

 QbImageSource is;
 is.type = qbiSource_AverageColor;

 /* run the get color subroutine */
 getColor(argv[1], is.average.Color);

 QbQueryCreate(&qhandle);
 QbQueryAddFeature(qhandle, "QbColorFeatureClass");
 QbQuerySetFeatureData(qhandle, "QbColorFeatureClass",&is);
 QbQuerySearch(qhandle, "ADS", "ADS_IMAGE", 10, 0, resultType
                             &count, results);
 for (int j = 0; j <count; j++) {
 printf(j,":\n");

 DBiBrowse("usr/local/bin/xv %s", MMDB_PLAY_HANDLE, handles[j],
                MMDB_PLAY_WAIT);
 }
```

Figure 5. An application that searches for images by content (Part 1 of 2)

```
QbQueryDelete(qhandle);

 SQLDisconnect(hdbc);
 SQLFreeConnect(hdbc);
 SQLFreeEnv(henv);
}
```

Figure 5. An application that searches for images by content (Part 2 of 2)

Operating environments

The DB2 extenders operate with DB2 Version 5.2 (or higher) in a client/server environment.

The supported client platforms are:

OS/2, AIX, Windows® 3.1, Windows NT®, Windows 95, Windows 98, Solaris Operating Environment, and HP-UX

The supported servers are:

OS/2, AIX, Windows NT, Solaris Operating Environment, and HP-UX

Figure 6 on page 12 shows the supported platforms.

The DB2 extenders can operate in a single-partition database environment.

EEE Only: The extenders can also operate in a multipartition database environment on the following platforms:

- AIX
- Solaris Operating Environment
- Windows NT

EEE Only: DB2 extenders must be used with DB2 Extended Enterprise Edition to operate in a multipartition database environment.

The minimum version and release levels required for the supported platforms are the same as those for DB2 Version 5.2.

Examples

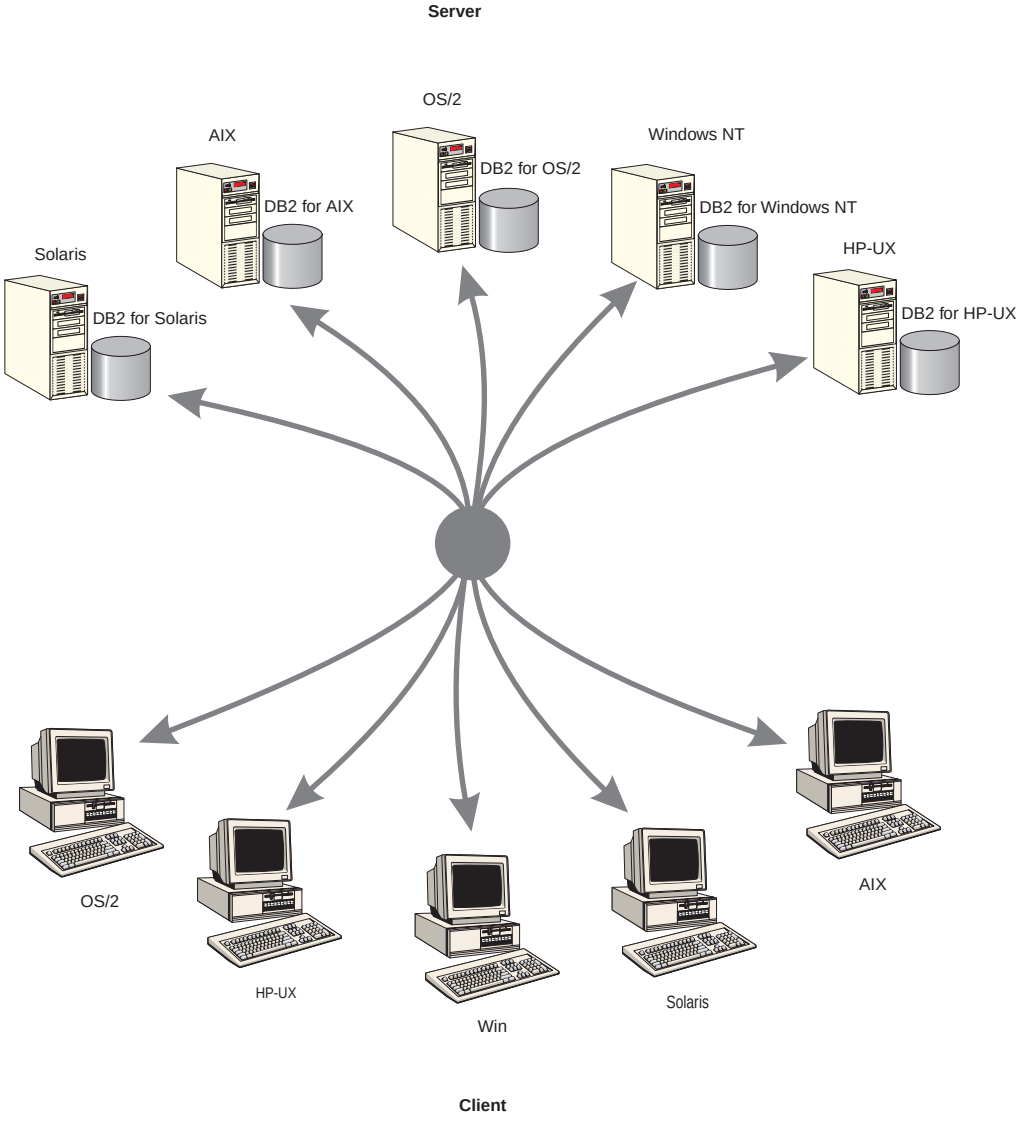


Figure 6. DB2 extender platforms

Chapter 2. DB2 Extender Concepts

This chapter describes concepts that you need to understand before using the DB2 extenders.

For more information about object-oriented concepts, see the *DB2 Embedded SQL Programming Guide.*

Object-oriented concepts

DB2 supports **object orientation**, the concept that anything, real or abstract, can be represented in an application as an object that comprises a set of operations and data values. For example, a document can be represented by a document object that comprises document data and operations that can be performed on the document, such as filing, sending, and printing. A video clip can be represented by a video object that comprises video data and operations such as playing the video clip or finding a specific video frame. Like real-world objects, representational objects have attributes. For example, a video object can be given attributes such as compression type and sampling rate.

Objects can be grouped together into types. Objects of the same type have the same attributes and behave in the same way, that is, they are associated with the same operations. For example, if a video type is defined to have a compression type attribute, then all objects of the video type have that attribute. If an object of the video type can be played, then all objects of the video type can be played.

DB2's support for object orientation allows you to store instances of object types in columns of tables, and operate on them by means of functions in SQL statements. For example, you can store video objects in a table column and operate on them using SQL functions. In addition, you can share the attributes and behavior of the stored objects among your applications. All the applications "see" the same set of attributes and behavior for the same object type.

Object-oriented concepts

Video objects are typically large and complex. So too are image and audio objects. As part of its support for object orientation, DB2 allows you to store large objects (LOBs) in a database. It also gives you ways to define and manipulate LOBs through user-defined types (UDTs), user-defined functions (UDFs), and triggers.

Large objects

DB2 allows you to store **large objects** (LOBs) in a database as:

- Binary large objects (BLOBs)
- Character large objects (CLOBs)
- Double-byte character large objects (DBCLOBs)

BLOBs are binary strings. Image, audio, and video objects are stored as BLOBs in a DB2 database. CLOBs are character strings made up of single-byte characters with an associated code page. This data type is used for text objects that contain single-byte characters. DBCLOBs are character strings made up of double-byte characters with an associated code page. This data type is used for text objects where double-byte characters are used.

Each LOB can be up to two gigabytes in length; however, DB2 allows many LOB columns per table. You can store up to 24 gigabytes of LOB space per row and up to 4 terabytes of LOB space per table.

Because of its size, a LOB's content is not directly stored in the user's table. Instead each LOB is identified in the table by a large object descriptor. The descriptor is used to access the large object stored elsewhere on the disk.

The DB2 extenders give you the added flexibility of keeping the content of a LOB in a file and pointing to it from the database. You make this designation when you use a DB2 extender to store an object.

User-defined types

Image, video, and audio objects are represented in the database as BLOBs. A **user-defined type** (UDT), also known as a **distinct type**, provides a way to differentiate one BLOB from another. For example, a UDT can be created for image objects and another for audio objects. Though stored as BLOBs, the image and audio objects are treated as types distinct from BLOBs and distinct from each other.

You create UDTs with an SQL CREATE DISTINCT TYPE statement. For example, suppose you are developing an application that processes geographic features on maps. You can create a distinct type named `map` for map objects as follows:

```
CREATE DISTINCT TYPE map AS BLOB (1M)
```

The map-type object is represented internally as a BLOB of 1 megabyte in length, but is treated as a distinct type of object.

You can use UDTs like SQL built-in types to describe the data stored in columns of tables. In the following example, a table is created with a column designed to hold map-type data:

```
CREATE TABLE places
  (locid     INTEGER NOT NULL,
   location  CHAR (50),
   grid      map)
```

Each DB2 extender creates a UDT for its type, that is, image, audio, and video.

User-defined functions

A **user-defined function** (UDF) is a way to create SQL functions and thus add to the set of built-in functions supplied with DB2. In particular, you can create UDFs that perform operations unique to image, audio, and video objects. For example, you can create UDFs to get the compression format of a video or return the sampling rate of an audio. This provides a way of defining the behavior of objects of a particular type. Video objects, for example, behave in terms of the functions created for the video type, and image objects behave in terms of the functions created for the image type.

You create UDFs with an SQL CREATE FUNCTION statement. The statement specifies, among other things, the data type to which the UDF can be applied. For example, the following statement creates a UDF named `map_scale` that calculates the scale of a map. Notice that the UDF identifies map as the data type to which it can be applied. The code that implements the function is written in C and is identified in the EXTERNAL NAME clause:

```
CREATE FUNCTION map_scale (map)
  RETURNS SMALLINT
  EXTERNAL NAME 'scale!map'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
```

UDFs can be used in an SQL statement in the same way as built-in functions. In the following example, the map_scale UDF is used in an SQL SELECT statement to return the scale of a map stored in a table column named grid:

```
SELECT map_scale (grid)
  FROM places
  WHERE location='SAN JOSE, CALIFORNIA'
```

Object-oriented concepts

Each DB2 extender creates a set of UDFs for its type, that is, image-specific, audio-specific, and video-specific UDFs. You use these UDFs in SQL statements to request extender functions such as storing an image in a table, getting the frame rate of a video, or adding comments about an audio.

UDF and UDT names

The full name of a DB2 function is *schema-name.function-name*, where *schema-name* is an identifier that provides a logical grouping for SQL objects. The schema name for DB2 extender UDFs is MMDBSYS. The MMDBSYS schema name is also the qualifier for the DB2 extender UDTs.

You can use the full name anywhere you refer to a UDF or a UDT. For example, MMDBSYS.CONTENT identifies a UDF whose schema name is MMDBSYS and whose function name is CONTENT. MMDBSYS.DB2IMAGE identifies a UDT whose schema is MMDBSYS and whose distinct-type name is DB2IMAGE. You can also omit the schema name when you refer to a UDF or UDT; in this case, DB2 uses the function path to determine the function or distinct data type that you want.

Function path

The **function path** is an ordered list of schema names. DB2 uses the order of schema names in the list to resolve references to functions and distinct data types. You can specify the function path by specifying the SQL statement SET CURRENT FUNCTION PATH. This sets the function path in the CURRENT FUNCTION PATH special register.

For the DB2 extenders, it is a good idea to add the `mmdbsys` schema to the function path. This allows you to enter DB2 extender UDF and UDT names without having to prefix them with mmdbsys. The following is an example of adding the mmdbsys schema to the function path:

```
SET CURRENT FUNCTION PATH = mmdbsys, CURRENT FUNCTION PATH
```

Do not add mmdbsys as the first schema in the function path if you log on as mmdbsys: If you log on with the mmdbsys user ID, the first schema in your function path is set to mmdbsys. If you then try to set the first schema in the function path to mmdbsys with a SET CURRENT FUNCTION PATH statement, your function path will begin with two mmdbsys schemas—an error condition.

Overloaded function names

Function names can be **overloaded**. This means that multiple UDFs, even in the same schema, can have the same name. However, two functions cannot have the same **signature**. A signature is the qualified function name concatenated with the defined data types of all the function parameters.

Triggers

A **trigger** defines a set of actions that are activated by a change to a table. Triggers can be used to perform actions such as validating input data, automatically generating a value for a newly inserted row, reading from other tables for cross-referencing purposes, or writing to other tables for auditing purposes. Triggers are often used for integrity checking or to enforce business rules.

You create a trigger using an SQL CREATE TRIGGER statement. The following statement creates a trigger to enforce a business rule regarding parts inventory. The trigger reorders a part when the number on hand is less than ten percent of the maximum number stocked.

```
CREATE TRIGGER reorder
       AFTER UPDATE OF on_hand, max_stocked ON parts
       REFERENCING NEW AS n_row
       FOR EACH ROW MODE DB2SQL

       WHEN (n_row.on_hand < 0.10 * n_row.max_stocked)
       BEGIN ATOMIC
              VALUES(issue_ship_request(n_row.max_stocked -
                                  n_row.on_hand,
                                  n_row.partno));
       END
```

The DB2 extenders create and maintain administrative support tables to record information about image, audio, and video data stored in a database. (See “Administrative support tables” for more information about these tables.) The extenders use triggers to update these tables when image, audio, or video data is inserted into, updated in, or deleted from a database.

Extender data structures

The Image, Audio, and Video Extenders create and use administrative support tables and handles to store and access image, audio, and video data. The Image Extender also creates and uses QBIC catalogs to access images by content. The Video Extender also uses index files and shot catalogs to access information about scene changes in a video.

Administrative support tables

Administrative support tables, also called metadata tables, contain the information that the extenders need to process user requests on image, audio, and video objects. The information in administrative support tables is often referred to as “metadata”.

Data structures

As Figure 7 on page 19 illustrates, some of the administrative support tables identify user tables and columns that are enabled for an extender. These tables reference other administrative support tables that are created to hold attribute information about objects in enabled columns. In these tables, the extenders maintain information about attributes that are unique to a particular extender-defined data type, as well as information about attributes that are common across extender data types. For example, the Image Extender maintains information about the width, height, and number of colors in an image, as well as information about attributes common to image, audio, and video objects, such as the identification of the person who imported the object into the database or who last updated the object.

The administrative support tables can also contain the contents of stored objects in BLOB format. Alternatively, an object can be kept in a file and referenced by the administrative support tables. For example, a video clip can be stored as a BLOB in an administrative support table or kept in a file that is referenced by the table.

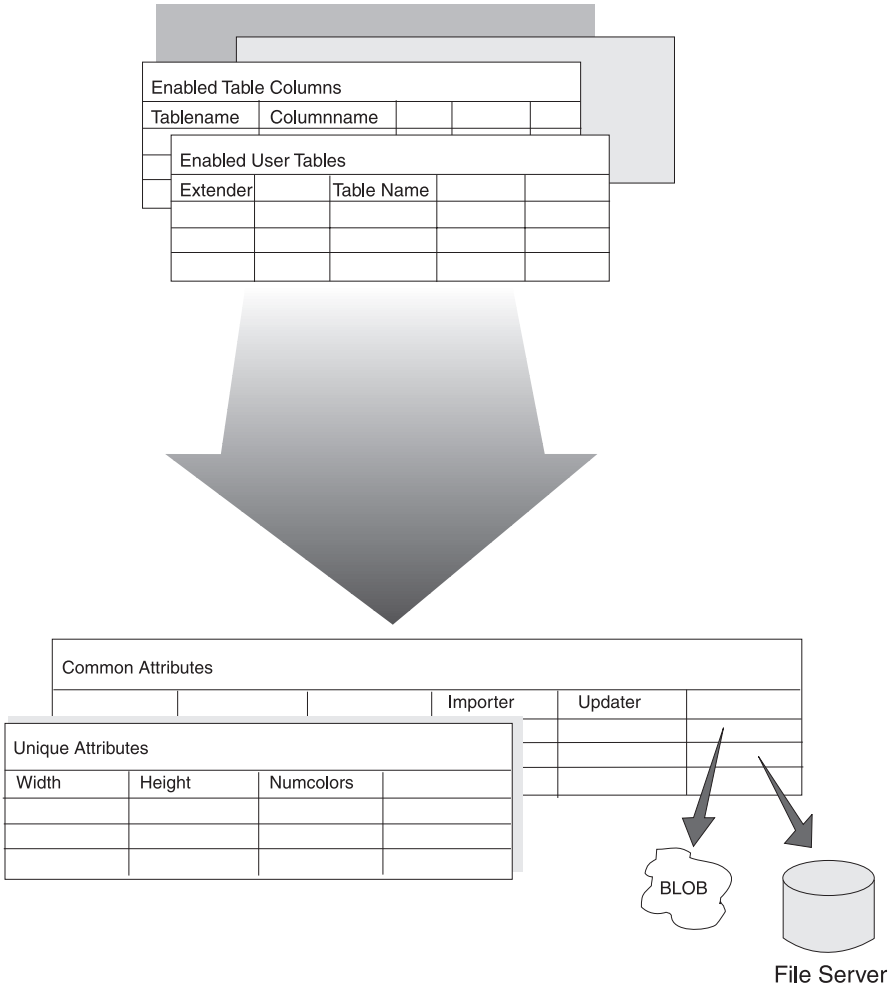


Figure 7. Administrative support tables

Handles

When you store an image, audio, or video object in a user table, the object is not actually stored in the table. Instead, an extender creates a character string called a **handle** to represent the object, and stores the handle in the table. The extender stores the object in an administrative support table, or stores a file identifier in an administrative support table if you keep the content of the object in a file. It also stores the object's attributes and handle in administrative support tables. In this way, the extender can link the handle stored in a user table with the object information stored in the administrative

support tables. Figure 8 illustrates the information stored for two images in a user table.

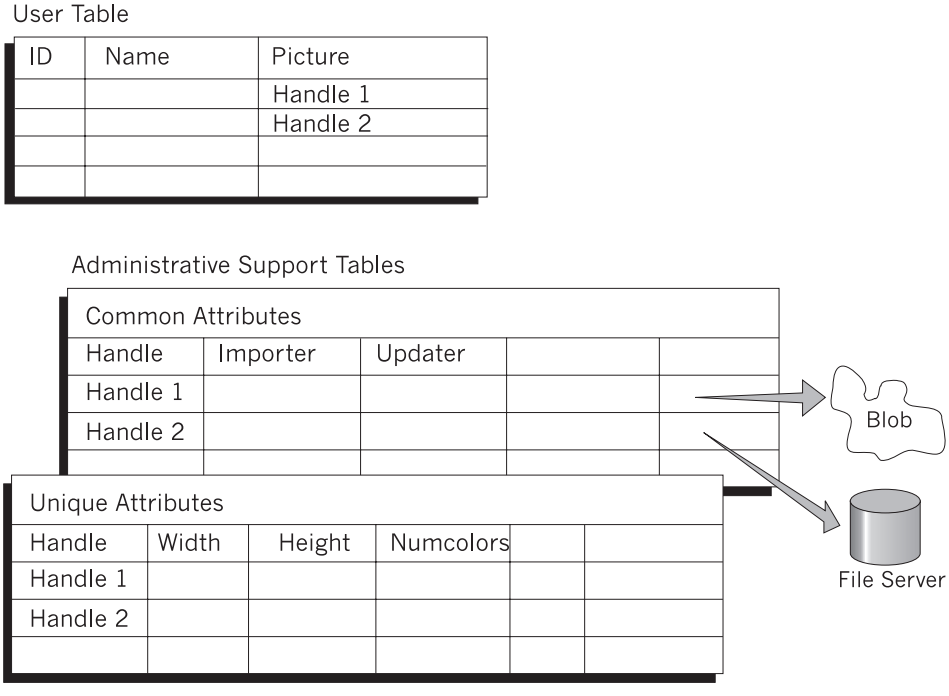


Figure 8. Handles

QBIC catalogs

A **QBIC catalog** is a set of files that hold data about the visual features of images. The Image Extender uses this data to search for images by content.

You create a QBIC catalog for each column of images in a user table that you want to make available for searching by content. When you create a QBIC catalog you identify the features for which you want the Image Extender to analyze, store, and later query data. You can also add or drop features from a QBIC catalog after the catalog is created.

A QBIC catalog can hold data for the following image features:

Average color The sum of the color values for all pixels in an image divided by the number of pixels in the image. (A pixel is the smallest element of an image that can be assigned color and intensity.) For example, if 50% of an image consists of blue pixels and the other 50% red pixels, the image has an average color value of purple. Average color is used to search for images that have

a predominant color. If an image has a predominant color, the average color will be similar to the predominant color.

Histogram color
Measures the distribution of colors in an image against a spectrum of 64 colors. For each of the 64 colors, histogram color identifies the percentage of pixels in an image that have that color. For example, the histogram color of an image might be 40% white pixels, 50% blue, and 10% red; none of the pixels in the image have any of the remaining colors in the histogram spectrum. Histogram color is used to search for images that have a variety of colors.

Positional color
The average color value for the pixels in a specified area in an image. For example, the upper right-hand corner of an image might show a bright yellow sun; the positional color of this area of the image is bright yellow. Positional color is used to search for images that have a predominant color in a particular area.

Texture Measures the coarseness, contrast, and directionality of an image. Coarseness indicates the size of repeating items in an image (for example, pebbles versus boulders). Contrast identifies the brightness variations in an image (light versus dark). Directionality indicates whether a direction predominates in an image (as in the vertical direction of a picket fence) or does not predominate (as in an image of sand). Texture is used to search for images that have a particular pattern.

To make an image available for searching by content, you catalog the image. When you catalog an image, the Image Extender analyzes the image, by computing the feature values for the image, and stores the values in a QBIC catalog.

When you search for an image by content, your query identifies a feature for the search (such as average color), a source for the feature (such as an example image), and a target set of cataloged images. The Image Extender computes the feature value of the source and compares it to the cataloged feature values for the target images. It then computes a score that indicates how similar the feature values of the target images are to the source.

You can have the Image Extender return the images whose features are most similar to the source. The Image Extender will return the handle of each

image and the image score. You can also have the Image Extender return only the score of a single image, or its rank (that is, how its score compares to the scores of other cataloged images).

Video indexes

A **video index** is a file that the Video Extender uses to find a specific shot or frame in a video clip.

The Video Extender can detect scene changes in a video. A **scene change** is a point in a video clip where there is a significant difference between two successive frames. This happens, for example, when a camera changes its point of view while recording a video. The frames between two scene changes constitute a **shot**.

You can use the Video Extender's scene detection capabilities to find a shot, or even an individual frame, in a video clip. To do this, the extender needs indexing information for the shot or frame. This indexing information is stored in an **index file**.

Shot catalogs

A **shot catalog** is used to store data about shots in a video clip. The shot catalog can be stored in a database or in a file.

A shot catalog stored in a file contains the following shot-related data:

- Shot catalog file name
- Values that control how the Video Extender detects a shot, for example, the mimimum number of frames in a shot
- Values that control how many frames and which frames will be stored as representative frames for a shot
- Shot number
- Starting frame number
- Ending frame number
- Representative frame number
- Name of a file that contains the contents of the representative frame

You can access the data in the shot catalog file or access a view of the shot catalog stored in a database. The view contains columns for the following shot-related data:

- Shot handle
- Video table name
- Video column names
- Video handle

- Video file name
- Starting frame number
- Ending frame number
- Representative frame number
- Representative frame data
- Comment

Partitioned database concepts (EEE only)

DB2 extenders can operate with DB2 Extended Enterprise Edition, and in this way take advantage of the partitioned database support provided by DB2 Extended Enterprise Edition.

A **partitioned database** is a database that is distributed across two or more independent machines. To the end-user and application developer, the database appears as a single database on a single machine. Partitioning allows applications to efficiently use a database that is simply too large to be handled by one machine.

A partitioned database is composed of two or more partitions. Each partition is managed by its own **database partition server**. A database partition server includes a database manager and the collection of data and system resources that it manages. Typically, one database partition server is assigned to each machine. However, it is possible to have multiple database partition servers on a single machine. Each database partition server holds a portion of the entire database. A database partition server is also sometimes called a **node**.

As Figure 9 on page 24 illustrates, database partitions can be grouped logically and assigned a name. Each group of database partitions is known as a **nodegroup**. Defining nodegroups allows you, for example, to limit application queries to selected database partitions, and thereby speed up transaction times. A nodegroup can contain one database partition only, or it can contain multiple database partitions. If a nodegroup contains multiple database partitions, it is known as a **multipartition nodegroup**. All database partitions named to a multipartition nodegroup must reside within the same database.

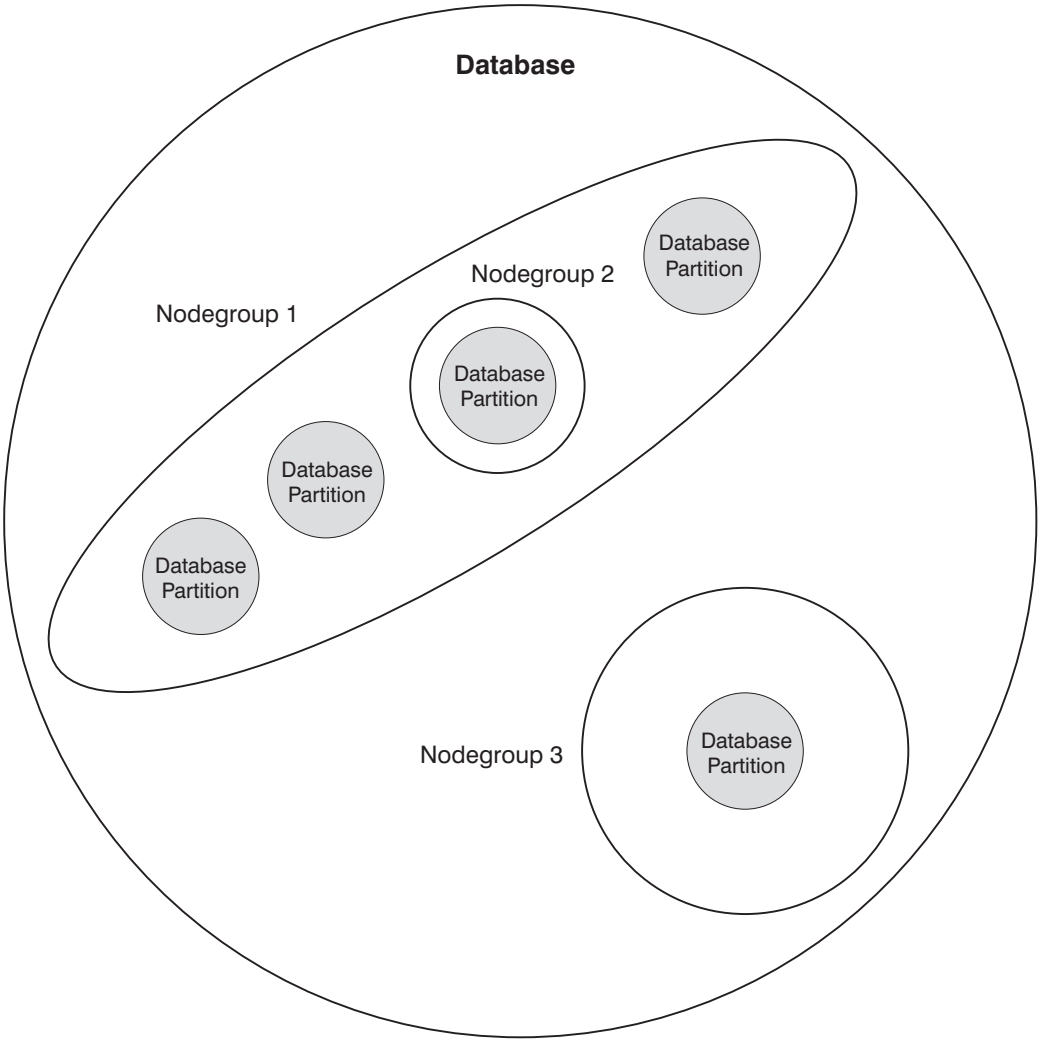


Figure 9. Nodegroups in a database

Using the extenders in a partitioned database system means you can:

- Reduce input/output and processing bottlenecks by distributing data across multiple partitions.
- Increase your database size by adding more machines and redistributing data across them.

Parallel processing

A partitioned database can use multiple CPUs to satisfy requests for information. Retrieval and update requests are automatically allocated into sub-requests and executed in parallel on the database partition servers on each machine.

As an illustration of the power of processing in a partitioned database system, assume that you have 100,000,000 records that you want to scan in a single-partition database. This scan would require that a single database manager search 100,000,000 records. Now suppose that these records are spread evenly over 20 database partition servers; each database manager only has to scan 5,000,000 records. If each database manager scans at the same time and with the same speed, the time required to complete the scan should be approximately 5% of that required of a single-partition system handling this task.

Scalability

As your database increases in size, you can add database partition servers to the database system to gain improved performance, a process known as **scaling** the database system.

When you scale a database, you add a database partition server, which in turn adds a database partition to each existing database in the database system. You can then assign the new database partition to an existing nodegroup for that database. Finally, you can redistribute data in that nodegroup to make use of the new database partition.

Using DB2 extenders in a partitioned database environment

By using DB2 extenders in a partitioned database environment, you can exploit features that support especially well the manipulation of LOBs. Large repositories of LOBs (which can be up to 2 gigabytes in length each) can be stored in one database, since a database can be spread across many machines.

Also, DB2 extenders participate in the parallel processing of SQL operations as managed by DB2 Extended Enterprise Edition. When DB2 Extended Enterprise Edition runs a query in parallel, any DB2 Extender UDF in the query is also run in parallel on the individual database partitions.

Security and recovery

Image, audio, and video objects stored as BLOBs in a DB2 database are afforded the same security and recovery protection as traditional numeric and character data. So too is the information stored for these objects in metadata tables. Users must have the required privilege to select, insert, or update objects.

```
GRANT SELECT FOR ALL ON employee TO ajones
```

In addition, some extender-related administrative operations require DBADM authority. See "Chapter 16. Application Programming Interfaces" on page 265 for the authority required by DB2 extender administrative APIs. See "Chapter 17. Administration Commands for the Client" on page 471 for the authority required by DB2 extender administrative commands.

When the content of an image, audio, or video is stored in a file referenced from the database, the metadata for the object is protected by DB2. The file must be in a directory that can be read by PUBLIC, that is, by all users.

BLOBs and metadata can be backed up and recovered in the same way as other data in DB2. Object contents stored in a file can be backed up and recovered using non-DB2 tools. Also, QBIC catalogs and video indexes can be backed up and recovered using non-DB tools. For information about backing up a QBIC catalog, see page 128. For information about backing up a video index, see page 178.

Chapter 3. How the Extenders Work

The DB2 extenders do a lot of work to handle image, audio, and video data requests so that you don't have to.

A good way to illustrate how the extenders work is to examine what they do when you use them. This chapter describes a scenario that includes the Image and Audio Extenders. It discusses the operations that users perform and how the extenders respond.

An extender scenario

The personnel department of a company wants to create a personnel database (in DB2 for AIX) that includes pictures of each employee.

A Database with pictures: As Figure 10 shows, an employee table in the database will contain the identification and name of each employee, as well as the employee's picture.

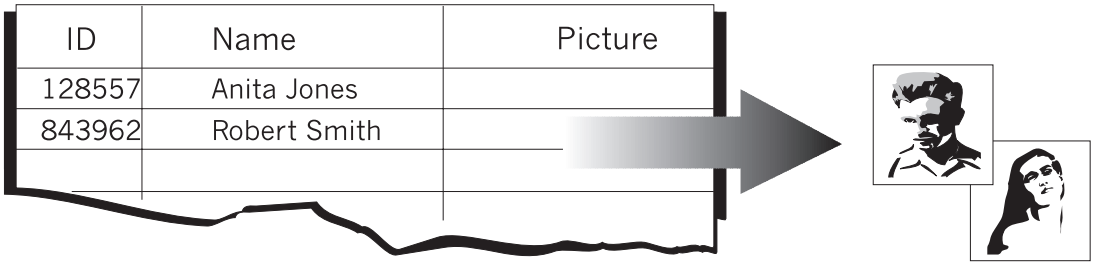


Figure 10. The employee table

To prepare the personnel database for image processing, a system administrator, that is, someone with SYSADM authority, begins by starting extender services. The system administrator then creates the database and enables it creates the database and enables the database server for use by the Image Extender.

A database administrator (DBA), or someone with equivalent authority, creates the employee table and then enables it and the employee picture column for use by the Image Extender.

A Database with sound: After the personnel database and employee table are prepared for image processing, the personnel department decides to add an

Scenario

audio recording for each employee to the table. This is shown in Figure 11 .

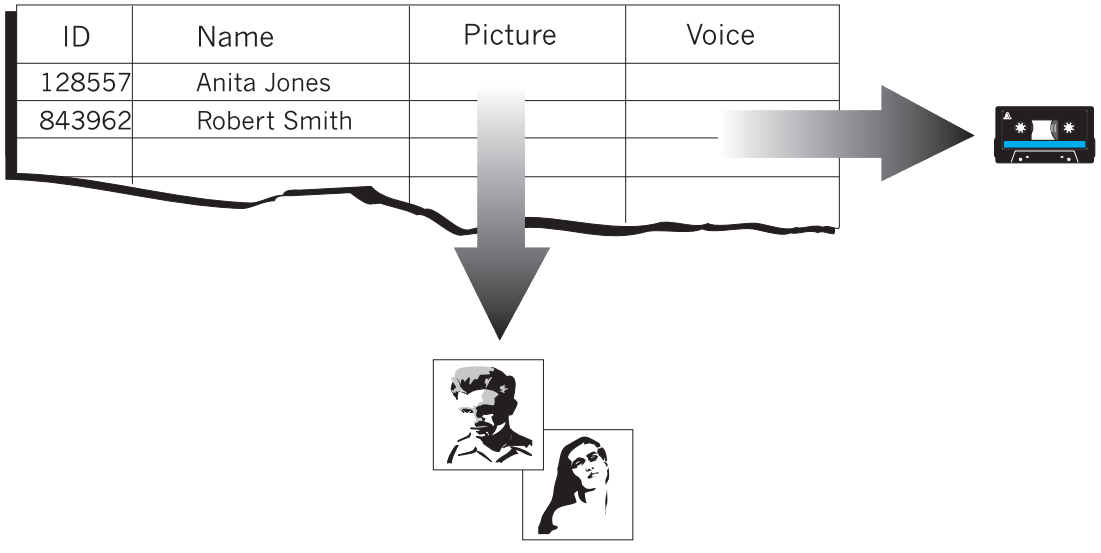


Figure 11. The employee table with an audio column added

The system administrator alters the table by adding a new column and enables the database, table, and column for use by the Audio Extender.

Users in the personnel department then insert data into, select and display data from, update data in, and delete data from the table.

Starting extender services

The extenders use services in the server as part of their operation. If these services are not already available as a function of normal “start-up” operations for the server, the system administrator starts them.

What the system administrator does: The system administrator logs on to the AIX server as the extender instance owner. The system administrator then issues the following command on the server:

```
DMBSTART
```

What happens: Extender services are started for the extender instance on the server. The DMBSTART command also starts a DB2 instance if it is not already running.

Preparing a database server

The system administrator creates and enablesthe personnel database for use by the Image Extender.

What the system administrator does: The system administrator creates the personnel database in DB2 for AIX using the following SQL statement:

```
CREATE DATABASE personnl          /*name of the database*/
       ON /persdb                 /*name of the database directory*/
       WITH "Personnel database"  /*comment*/
```

The system administrator connects to the database and enables it for use by the Image Extender. The system administrator uses the db2ext command-line processor to issue the following commands:

```
CONNECT TO personnl
ENABLE DATABASE FOR DB2IMAGE
```

What happens: In response to the ENABLE DATABASE command, the Image Extender:

- Creates a user-defined type named DB2IMAGE for image objects.
- Creates administrative support tables for image objects.
- Creates user-defined functions for image objects. The UDFs are listed in Table 1.

Table 1. User-defined functions created by the Image Extender

UDF name	Description
Comment	Get or update user comments
Content	Get or update the content of an image
DB2Image	Store the content of an image
Filename	Get the name of the file that contains an image
Format	Get the image format (for example, GIF)
Height	Get the height of an image in pixels
Importer	Get the user ID of the importer the of an image
ImportTime	Get the timestamp when an image was imported
NumColors	Get the number of colors used in an image
QbScoreFromName	Get the similarity score of an image (using a named query)
QbScoreFromStr	Get the similarity score of an image (using a query string)
QbScoreTBFromName	Get a table of similarity scores for a column of images (using a named query)
QbScoreTBFromStr	Get a table of similarity scores for a column of images (using a query string)

Table 1. User-defined functions created by the Image Extender (continued)

UDF name	Description
Replace	Update the content and user comments for an image
Size	Get the size of an image in bytes
Thumbnail	Get a thumbnail-sized version of an image
Updater	Get the user ID of the updater of an image
UpdateTime	Get the timestamp when an image was updated
Width	Get the width of an image in pixels

Preparing a table

The DBA creates the employee table and enables it and the picture column for use by the Image Extender.

What the DBA does: For convenience, the DBA adds the mmdbsys schema in the current functionpath using the following SQL statement:

```
SET CURRENT FUNCTION PATH = mmdbsys, CURRENT FUNCTION PATH
```

This allows UDT and UDF names to be specified without having to prefix them with the mmdbsys schema name. (The mmdbsys schema does not have to be the first schema in the function path.) See “UDF and UDT names” on page 16 for more information about UDT and UDF names.

The DBA creates the employee table. The DBA uses the DB2 command-line processor to issue the following SQL statement:

```
CREATE TABLE   employee           /*name of the table*/
       (id      CHAR(6)           /*employee identification*/
       name    VARCHAR(40)        /*employee name*/
       picture DB2IMAGE)          /*employee picture*/
```

The DBA then uses the db2ext command-line processor to issue the following commands:

```
ENABLE TABLE employee FOR DB2IMAGE
ENABLE COLUMN employee picture FOR DB2IMAGE
```

What happens: In response to the ENABLE TABLE command, the Image Extender:

- Identifies the employee table for use.
- Creates administrative support tables that hold attribute information for image objects in enabled columns.

In response to the ENABLE COLUMN command, the Image Extender:

- Identifies the picture column for use.
- Creates triggers. These triggers update various administrative support tables in response to insert, update, and delete operations on the employee table.

Altering a table

The DBA adds an audio column to the employee table and enables it for use by the Audio Extender.

What the DBA does: The DBA uses the db2ext command-line processor to enable the personnel database for use by the Audio Extender:

```
ENABLE DATABASE FOR DB2AUDIO
```

The DBA then uses the DB2 command-line processor to alter the employee table using the following SQL statement:

```
ALTER TABLE employee              /*name of the table*/
      ADD voice DB2AUDIO          /*employee audio recording*/
```

The DBA uses the db2ext command-line processor to enable the employee table and the voice column for use by the Audio Extender:

```
ENABLE TABLE employee FOR DB2AUDIO
ENABLE COLUMN employee voice FOR DB2AUDIO
```

What happens: In response to the ENABLE DATABASE command, the Audio Extender:

- Creates a user-defined type named DB2AUDIO for audio objects.
- Creates administrative support tables for audio objects.
- Creates user-defined functions for audio objects. The UDFs are listed in Table 2.

Table 2. User-defined functions created by the Audio Extender

UDF name	Description
AlignValue	Get the bytes per sample value of the audio
BitsPerSample	Get the number of bits used to represent the audio
BytesPerSec	Get the average number of bytes per second of audio
Comment	Get or update user comments
Content	Get or update the content of an audio
DB2Audio	Store the content of an audio
Duration	Get the playing time of an audio

Table 2. User-defined functions created by the Audio Extender (continued)

UDF name	Description
Filename	Get the name of the file that contains an audio
FindInstrument	Get the number of the audio track that records a specific instrument in an audio
FindTrackName	Get the track number of a named track in an audio recording
Format	Get the audio format
GetInstruments	Get the names of the instruments recorded in an audio
GetTrackNames	Get the track names in an audio
Importer	Get the user ID of the importer of an audio
ImportTime	Get the timestamp when an audio was imported
NumAudioTracks	Get the number of recorded tracks in an audio
NumChannels	Get the number of audio channels
Replace	Update the content and user comments for an audio recording
SamplingRate	Get the sampling rate of the audio
Size	Get the size of an audio in bytes
TicksPerQNote	Get the number of clock ticks per quarter note used in recording an audio
TicksPerSec	Get the number of clock ticks per second used in recording an audio
Updater	Get the user ID of the updater of an audio
UpdateTime	Get the timestamp when an audio was updated

In response to the ENABLE TABLE command, the Audio Extender

- Identifies the employee table for use.
- Creates administrative support tables that hold attribute information for audio objects in enabled columns.

In response to the ENABLE COLUMN command, the Audio Extender:

- Identifies the voice column for use.
- Creates triggers. These triggers update various administrative support tables in response to insert, update, and delete operations on the employee table.

Inserting data into a table

A user inserts a record for Anita Jones into the employee table. The record includes Anita's identification (128557), name, picture, and voice recording. The source image and audio content are in files on the server. The image is stored in the table as a BLOB; the content of the audio remains in the server file (and is referenced from the table).

What the user does: The user inserts the record into the employee table using an application program that includes the statements shown in Figure 12.

```
EXEC SQL BEGIN DECLARE SECTION;
long hvInt_Stor;
long hvExt_Stor;
EXEC SQL END DECLARE SECTION;

hvInt_Stor = MMDB_STORAGE_TYPE_INTERNAL;
hvExt_Stor = MMDB_STORAGE_TYPE_EXTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',                          /*id*/
        'Anita Jones',                     /*name*/
        DB2IMAGE(                          /*Image Extender UDF*/
           CURRENT SERVER,                 /*database server name in*/
                                           /CURRENT SERVER register*/
           '/employee/images/ajones.bmp'   /*image source file*/
           'ASIS',                         /*keep the image format*/
           :hvInt_Stor,                    /*store image in DB as BLOB*/
           'Anita''s picture'),            /*comment*/
        DB2AUDIO(                          /*Audio Extender UDF*/
           CURRENT SERVER,                 /*database server name in*/
                                           /*CURRENT SERVER register*/
           '/employee/sounds/ajones.wav',  /*audio source file*/
           'WAVE',                         /* audio format */
           :hvExt_Stor,                    /*retain content in server file*/
           'Anita''s voice')               /*comment*/
      );
```

Figure 12. Inserting data into a table

What happens In response to the DB2Image UDF in the INSERT statement, the Image Extender:

- Reads the attributes of the image, such as its height, width, and number of colors, from the source image file header.
- Creates a unique handle for the image, and records in an administrative support table:
  - The handle for the image

  - A timestamp
  - The image size in bytes
  - The comment “Anita’s picture”
  - The content of the image

  The image source is in a server file named ajones.bmp. The content of the file is inserted into the administrative support table record as a BLOB. The format of the stored image is the same as the source image, that is, no format conversion is done.
- Stores a record in an administrative support table. The record contains image-specific attributes, such as the number of colors in the image, as well as a thumbnail-sized version of the image.

In response to the DB2Audio UDF in the INSERT statement, the Audio Extender:

- Reads the attributes of the audio, such as the number of audio tracks and channels, from the source audio file header.
- Creates a unique handle for the audio
- Stores a record in an administrative support table The record contains:
  - The handle for the audio
  - A timestamp
  - The audio size in bytes
  - The comment “Anita’s voice”

  The audio content is in a server file named ajones.wav; the file is referenced by the administrative support table record.
- Stores a record in another administrative support table. The record contains audio-specific attributes such as the sampling rate of the audio.

Triggers are invoked to insert the image and audio attribute data into various administrative support tables.

Selecting data from a table

A user retrieves information about how recently Robert Smith’s image and voice recording were stored in the employee table.

What the user does: The user gets the information using an application program that includes the SQL statements shown in Figure 13 on page 35.

```
EXEC SQL BEGIN DECLARE SECTION;
char[255]  hvImg_Time;
char[255]  hvAud_Time;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT IMPORTTIME(PICTURE),              /*when image was stored*/
       IMPORTTIME(VOICE)                          /*when audio was stored*/
       INTO :hvImg_Time, :hvAud_Time
       FROM EMPLOYEE
       WHERE NAME='Robert Smith';
```

Figure 13. Selecting data from a table

What happens: In response to the ImportTime UDF for the PICTURE column, the Image Extender returns a timestamp that contains the date and time that the image was stored. In response to the ImportTime UDF for the VOICE column, the Audio Extender returns a timestamp that contains the date and time that the voice recording was stored.

Displaying and playing objects

A user displays Robert Smith's image and plays Robert Smith's voice recording. The image is stored in the employee table as a BLOB; the content for the voice recording is in a server file.

What the user does: The user displays the image and plays the voice recording using an application program that includes the SQL statements shown in Figure 14 on page 36.

Displaying/playing objects

```
EXEC SQL BEGIN DECLARE SECTION;
char hvImg_hdl [251];
char hvAud_hdl [251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT PICTURE,                                /*Get image handle*/
                VOICE                                   /*Get audio handle*/
       INTO :hvImg_hdl, :hvAud_hdl
       FROM EMPLOYEE
       WHERE NAME='Robert Smith';

rc=DBiBrowse(
        NULL,                                           /*Use default image browser*/
        MMDB_PLAY_HANDLE,                               /*Use handle*/
        hvImg_hdl,                                      /*Image handle*/
        MMDB_PLAY_NO_WAIT);                             /*Run browser independently*/

rc=DBaPlay(
        NULL,                                           /*Use default audio player*/
        MMDB_PLAY_HANDLE,                               /*Use handle*/
        hvAud_hdl,                                      /*Audio handle*/
        MMDB_PLAY_WAIT);                                /*Wait for player to end*/
                                                        /*before continuing*/
```

Figure 14. Displaying and playing objects

What happens: DB2 retrieves the handle of Robert Smith's image and voice recording. Then, in response to the DBiBrowse API, the Image Extender gets the image content associated with the retrieved image handle. The Image Extender retrieves the image content from the database and puts it into a temporary client file for display by an image browser. The NULL parameter indicates that the default image browser for the user's system will be used. The browser will run independently of the calling program, meaning that the calling program will not wait for the image browser to finish before continuing.

In response to the DBaPlay API, the Audio Extender gets the file name of the audio associated with the retrieved audio handle and passes the file name to the audio player. The NULL parameter indicates that the default audio player for the user's system will be used. The calling program will wait for the user to end the audio player before continuing.

Updating data in a table

Anita Jones replaces her picture in the employee table with a more recent picture. The content of the newer picture is in a server file.

What the user does: The user replaces the picture in the employee table using an application program that includes the SQL statements shown in Figure 15.

```
EXEC SQL BEGIN DECLARE SECTION;
  char hvComment [16385];
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

strcpy(hvComment, "Picture taken at Anita's promotion");
hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

EXEC SQL UPDATE EMPLOYEE
       SET PICTURE=REPLACE(
              PICTURE,                      /*image handle*/
              '/myimages/newone.bmp',       /*source image content*/
              'BMP',                        /*source format*/
              :hvStorageType,               /*store image in table as BLOB*/
              :hvComment)                   /*replace comment*/
       WHERE NAME='Anita Jones';
```

Figure 15. Updating data in a table

What happens: In response to the Replace UDF in the UPDATE statement, the Image Extender reads the attributes of the new image and uses them to update the attributes stored in the administrative support tables for the old image. The image source is in a server file named newone.bmp. The content of the file is inserted into the administrative support table record as a BLOB, replacing the BLOB content of the old image.

Triggers are invoked to replace the image attribute data in various administrative support tables.

Deleting data from a table

A user deletes Anita Jones's record from the employee table.

What the user does: The user deletes the record from the employee table using an application program that includes the following SQL statement:

```
DELETE FROM EMPLOYEE
       WHERE NAME='Anita Jones';
```

What happens: Triggers delete entries for Anita Jones in various administrative support tables.

Part 2. Administering Image, Audio, and Video Data

Chapter 4. Administration Overview

This chapter provides an overview of the administration tasks involved when you create applications using the DB2 Extenders.

The DB2 extenders offer two ways to perform most administration tasks:

- Administration application programming interfaces. You can include the DB2 extenders APIs in your C language program. See “Chapter 16. Application Programming Interfaces” on page 265 for reference information on these APIs.
- Administration commands. You can submit administration commands to the db2ext command-line processor. These commands do not run on the DB2 command line. See “Chapter 17. Administration Commands for the Client” on page 471 and “Chapter 18. Administration Commands for the Server” on page 511 for instructions on entering administration commands and for additional reference information.

Administration tasks you can perform with the DB2 extenders

There are five categories of administrative tasks:

- Managing extender services. The DB2 extenders run on their own servers on top of DB2. Before applications can use extender data, the system administrator starts the extender services and the user connects to the database that holds the extender data.
- Preparing data objects for extender data. You prepare databases, tables, and columns to hold extender data by enabling them. When you enable a data object, the extenders create and maintain administrative support tables (also called metadata tables) to manage the extender data.
- **EEE only.** Redistributing extender data in a partitioned environment. When you add or drop partitions from a partitioned database, you can redistribute data to take advantage of the new node configuration.
- Tracking data objects and media files. As you debug applications that use the DB2 extenders, it’s useful to know which data objects are enabled for extender data. It’s also useful to understand the correlation between user tables and external media files.
- Cleaning up administrative support files. As you work with the DB2 extenders, obsolete entries can eventually accumulate in the administrative support tables. Deleting the obsolete metadata can improve performance and reclaim storage space.

Administration overview

Table 3 lists all the tasks involved in administering extender data. The table specifies which tools are provided to perform each task and where to find more information.

In the **Extender API** column, the third character of each API statement is represented by x. This character varies according to the extender you are using:

Character	Extender
a	Audio
i	Image
v	Video

For example, the API for enabling a table for image data is DB**i**EnableTable, the API for enabling a table for audio is DB**a**EnableTable, and the API for enabling a table for video is DB**v**EnableTable. A value of No in the Extender API column means that there is no extender API for the task. A value of No in the Extender Command column means that there is no extender command for the task.

QBIC requires additional administration: If you plan to use the Image Extender's Query by Image Content (QBIC) capability, you need to perform additional administrative tasks, such as creating a QBIC catalog. For information about these tasks, see "Chapter 13. Querying Images by Content" on page 125.

Table 3. Administration tasks and facilities for the DB2 extenders

Task	Extender API	Extender Command	See
Managing extender services			
Start the extender services	No	DMBSTART	p. 47
Get status of the extender services	No	DMBSTAT	p. 50
Stop the extender services	No	DMBSTOP	p. 49
Connect to a database	No	CONNECT	p. 47
Start an extender service for your database	No	START SERVER	p. 49
Get status of an extender service for your database	No	GET SERVER STATUS	p. 50
Stop an extender service for your database	No	STOP SERVER	p. 49
Preparing data objects for multimedia data			
Enable a database	DBxEnableDatabase	ENABLE DATABASE	p. 51

Table 3. Administration tasks and facilities for the DB2 extenders (continued)

Task	Extender API	Extender Command	See
Disable a database	DBxDisableDatabase	DISABLE DATABASE	p. 55
Enable a table	DBxEnableTable	ENABLE TABLE	p. 53
Disable a table	DBxDisableTable	DISABLE TABLE	p. 55
Enable a column	DBxEnableColumn	ENABLE COLUMN	p. 54
Disable a column	DBxDisableColumn	DISABLE COLUMN	p. 55
Redistributing extender data in a partitioned environment (EEE only)			
Redistribute extender data based on a new nodegroup configuration	DMBRedistribute	REDISTRIBUTE NODEGROUP	p. 57
Tracking data objects and media files			
Find out if databases are enabled	DBxIsDatabaseEnabled	GET EXTENDER STATUS	p. 59
Find out if tables are enabled	DBxIsTableEnabled	GET EXTENDER STATUS	p. 59
Find out if columns are enabled	DBxIsColumnEnabled	GET EXTENDER STATUS	p. 59
Find table entries that reference files in tables whose qualifier is the current user ID	DBxIsFileReferenced	No	p. 60
Find table entries that reference files in all tables of a specific qualifier or all tables in a database	DBxAdminIsFileReferenced	No	p. 60
Find files referenced by table entries in tables whose qualifier is the current user ID	DBxGetReferencedFiles	GET REFERENCED FILES	p. 61
Find files referenced by table entries in all tables of a specific qualifier or all tables in a database	DBxAdminGetReferencedFiles	GET REFERENCED FILES	p. 61
Find inaccessible files referenced by table entries in all tables whose qualifier is the current user ID	DBxGetInaccessibleFiles	GET INACCESSIBLE FILES	p. 63

Table 3. Administration tasks and facilities for the DB2 extenders (continued)

Task	Extender API	Extender Command	See
Find inaccessible files referenced by table entries in all tables of a specific qualifier or all tables in a database	DBxAdminGetInaccessibleFiles	GET INACCESSIBLE FILES	p. 63
Cleaning up administrative support (metadata) tables			
Clean up metadata tables for a specific user table or all user tables whose qualifier is the current user ID	DBxReorgMetadata	REORG	p. 65
Clean up metadata tables for all user tables with a specific qualifier or all user tables in a database	DBxAdminReorgMetadata	REORG	p. 65

Sequence of administration tasks: The following list is an ordered summary of the administration tasks you perform when you use the extenders the first time. You perform some tasks using DB2 commands or statements, the others with the DB2 extenders. This sequence assumes your DB2 system is running.

Required tasks:

1. Start the extender services.
2. Create a database (using DB2).
3. Connect to the database server.
4. Enable the database.
5. Create a table and column (using DB2).
6. Enable a table in the database.
7. Enable a column in the table.

Optional tasks:

1. Track data objects and media files.
2. Set the function path (using DB2).
3. Clean up administrative support files.

Examples: Most of the examples in the next four chapters assume a system administrator (SYSADM) or a database administrator (DBA) is performing the tasks. A few tasks do not require DBA or SYSADM authority.

The examples assume that the DBA has added the MMDBSYS schema in the current function path. This allows the DBA to specify UDT names without prefixing them with the MMDBSYS schema name. For more information about UDT names, see “UDF and UDT names” on page 16.

Many of the API examples in this section are based on the sample application code supplied with extenders. The sample code is in the SAMPLES subdirectory on the client.

Administration overview

Chapter 5. Managing Extender Servers

The DB2 Extenders run in the DB2 client/server environment. This environment consists of a database server and one or more remote database clients. The DB2 extender services run on the server. Before you can access them, you have to start them.

After the environment is set up, you can stop and restart extender services from the client. From either the client or the server, you can get the status of the extenders.

EEE Only: In a multipartition environment, you can also add and drop database partitions.

Establishing the extender environments

From the operating system command line on the server, enter the DMBSTART command to start the extender services:

```
dmbstart
```

The DMBSTART command starts extender services for all databases that are enabled to hold extender data. The command also starts DB2 if it is not running. You need SYSADM, SYSCTRL, or SYSMAINT authority to run the command. On AIX, you must be logged on as the extender instance owner.

At this point, your C language application can access extender services through the APIs, if the application establishes a connection to the database. Similarly, if you want to use the db2ext command line, you must connect to the database you want to work with. The db2ext command line requires an independent connection separate from one used by the DB2 command line.

Open the db2ext command-line processor on the client and run the DB2 extenders CONNECT command. In the following example, the command connects to the PERSONNL database. It accesses tables with the ANITAS qualifier using the ANPASS password:

```
connect to personnl user anitas using anpass
```

EEE Only: If you are using DB2 extenders in a partitioned database environment, the DMBSTART command will start extender services in all database partition servers defined for the instance. If you want to start extender services at one database partition server only, you do so by

specifying in the command the node you want to start. The example below shows what you would type to start extender services at node number 2.

```
dmbstart nodenum 2
```

EEE Only: Before you start a single database partition server, you must start DB2 at that node.

Now you can run the rest of the DB2 extender commands listed in "Chapter 17. Administration Commands for the Client" on page 471.

Adding and dropping database partitions (EEE only)

In order to use extenders in a partitioned database environment, the partitions defined for the extenders must match those defined for DB2. The DMBSTART command starts the extender servers on each of the nodes defined for the current instance. The server will automatically detect if the node it is running on has been recently created and perform any necessary initialization. If a node is dropped from DB2, the extender files associated with that node must be manually deleted.

For more information on DB2 commands for adding and dropping partitions, refer to *IBM DB2 Universal Database Extended Enterprise Edition Quick Beginnings*

The following steps are required to add a database partition:

1. Create a partition for DB2 using the command DB2NCRT or the command DB2START ADDNODE.
2. Create a partition for the extenders using the extender command DMBSTART NODENUM.
3. Redistribute DB2 data to take advantage of the new node configuration using the DB2 command REDISTRIBUTE NODEGROUP.
4. Redistribute extender data to take advantage of the new node configuration using the extender command REDISTRIBUTE NODEGROUP.

The following steps are required to drop a database partition:

1. Redistribute DB2 data to remove it from the partition you want to drop using the DB2 command REDISTRIBUTE NODEGROUP.
2. Redistribute extender data to remove it from the partition you want to drop using the extender command REDISTRIBUTE NODEGROUP.
3. Drop a partition for DB2 by using the DB2 command DB2NDROP or the command DB2STOP DROP.

4. Drop a partition for the extenders using the extenders command DMBSTART NODENUM.
5. Manually remove the extender files associated with the dropped partition.

The extender data for a database partition is in a subdirectory named node*num*, where *num* is the node number corresponding to the database partition. The subdirectory is in a directory that is specified by the value of the DB2MMDATAPATH environment variable. To remove extender data for a dropped database partition, delete the appropriate node*num* subdirectory and all subdirectories below it. (For more information about DB2MMDATAPATH, see "How the DB2MMDATAPATH environment variable is used (EEE only)" on page 553.)

Stopping and starting extender servers

When you stop applications that use extender services, the server remains active until you explicitly stop it, or until the server machine recycles. You can stop all the extender servers by entering the DMBSTOP command from the server machine on the command line for the operating system.

To stop and restart extender services from the client, run the STOP SERVER and START SERVER commands from the db2ext command line. These commands stop and start extender services for the current database.

EEE Only: In a partitioned database environment, DMBSTART can be used to start either all database partition servers defined for the instance, or a single database partition server only. DMBSTART without any parameters will start all database partition servers. If you want to start one database partition server only, you do so by specifying in the command the node you want to start, as shown below:

```
dmbstart nodenum 2
```

Once you have started the server at a particular node, you must then reconnect that server to the database using the extender command RECONNECT SERVER, as shown below:

```
reconnect server at nodenum 2
```

EEE Only: Likewise, if you are using DB2 extenders in a partitioned database environment, DMBSTOP without any parameters will stop all database partition servers defined for the instance. If you want to stop one database partition server only, you must first disconnect that server from the database using the extenders command DISCONNECT SERVER, as shown below:

```
disconnect server at nodenum 2
```

You can then run DMBSTOP, specifying in the command the node you want to stop. The example below shows what you would enter from the command line of the server to stop extender services at node number 2.

```
dmbstop nodenum 2
```

EEE Only: Do not run DMBSTOP at a specific node unless your database is running in maintenance mode. In addition, you need to make sure that no extender activities will be triggered on this node while it is turned off. Otherwise, you may encounter unexpected behavior.

Displaying server status

From the server, you can display the extender server status with the DMBSTAT command. For example, the following command lists the databases that are enabled and whether the extenders are up and running. Connect to the server before running this command.

```
dmbstat
```

From the client, you can get the status of the extender server for a database using the GET SERVER STATUS command. For example, the following command lists the status of the personnl database:

```
get server status personnl
```

Chapter 6. Preparing Data Objects for Extender Data

You prepare databases, tables, and columns to hold extender data by enabling them. First enable the database. Then enable a table in the database. Finally, enable a column in the table.

When you no longer want extender data in your data objects, you can disable the objects.

You can enable and disable objects either using the APIs in your C language program or from the db2ext command line. In this chapter, examples are provided for each method.

Enabling databases

When you enable a database, the extender:

- Creates a user-defined type (UDT) named DB2xxxxx for your data objects, where xxxxx is either `Image`, `Audio`, or `Video`. The UDT is used to define a column in the user table that holds handles for objects of the of that type.
- Creates administrative support tables (also called metadata tables) for the database. These tables are not user tables (tables in which users store business data). They are used by the extenders to manage extender data. Do not edit them manually.
- Creates the user-defined functions (UDFs) associated with the extender. The UDFs are listed in "User-Defined Functions" on page 197.

When enabling a database, you must also specify tablespaces to hold the administrative support tables (and their indexes) for the database. One or more of the tablespaces specified can be a null value, in which case a default tablespace is used.

You need DBA authority to enable a database.

EEE Only: When enabling a database for an extender in a partitioned environment, the tablespace you specify should be defined in a nodegroup that includes all the nodes in the partitioned database system. Also, the tablespace specified should be located in the same nodegroup as the user table.

In the following examples, a database is enabled to hold image data using the default table space.

Using the API: The code in Figure 16 connects to an existing database before enabling it. This example is written using the DB2 call level interface. It includes some set-up and error-checking code. The complete sample program is in the ENABLE.C file in the SAMPLES subdirectory.

```
/*---- Set-up -----------------------------------------------------*/
#include <stdio.h>
#include <stlib.h>
#include <string.h>
#include "dmbimage.h"       /* image extender function prototypes (DBi) */
#include "utility.h"        /* utility functions */

#define MMDB_ERROR_MSG_TEXT_LEN         1000

int
main(int argc, char *argv[])
{
        SQLHENV henv = SQL_NULL_HENV;
        SQLHDBC hdbc = SQL_NULL_HDBC;
        SQLHSTMT hstmt = SQL_NULL_HSTMT;
        SQLCHAR *uid, *pwd;
        SQLCHAR server[SQL_MAX_DSN_LENGTH];
        SQLCHAR buffer[500];
        SQLCHAR szCreate[] = "CREATE TABLE %s (%s mmdbsys.DB2Image,
                   name varchar(25), address varchar(25), jobtitle char(11),
                   dept char(10), salary char(10))";
        SQLRETURN rc = SQL_SUCCESS;
        SQLINTEGER sqlcode = 0;
        char errorMsgText[MMDB_ERROR_MSG_TEXT_LEN+1];
        char dbName[18+1];
        char tableName[8+18+1] = "employee";
        char imageColumn[18+1] = "picture";
        char *step;
        if (argc < 3) {
                printf("Usage: %s <dbname> <userid> <password>\n", argv[0]);
                exit(1);
        }
        else {
                strcpy(dbName, argv[1]);
                uid = (SQLCHAR*) argv[2];
                pwd = (SQLCHAR*) argv[3];
        }
```

Figure 16. Sample code that enables a database (Part 1 of 2)

```
/*---- Connect to the database. -----------------------------------*/
strcpy((char*) server, dbName);
rc = cliInitialize(&henv, &hdbc, server, uid, pwd);
cliCheckError(henv, hdbc, SQL_NULL_HSTMT, rc);
if (rc < 0) goto ERROR;

/****** Enable the database. ****************************************/
printf("%s: Enabling database......\n", argv[0]);
printf("%s: This may take a few minutes, please wait......\n", argv[0]);
step="DBiEnableDatabase with NULL tablespace";
rc = DBiEnableDatabase(NULL);
if (rc < 0) {
        printf("%s: %s failed!\n", argv[0], step);
        printMsg(rc);
        DBiGetError(&sqlcode, errorMsgText);
        printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);} else if (rc > 0) {
        printf("%s: %s, warning detected.\n", argv[0], step);
        printMsg(rc);
        DBiGetError(&sqlcode, errorMsgText);
        printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else
        printf("%s: %s OK\n", argv[0], step);
```

Figure 16. Sample code that enables a database (Part 2 of 2)

Using the db2ext command line: In this example, the database is already connected.

```
enable database for db2image
```

Enabling tables

When enabling a user table, you must also specify tablespaces to hold the administrative support tables (and their indexes) that go with it. One or more of the tablespaces specified can be a null value, in which case a default tablespace is used.

EEE Only: When enabling a table for an extender in a partitioned environment, the tablespace you specify should be defined in a nodegroup that includes all the nodes in the partitioned database system. Also, the tablespace specified must be located in the same nodegroup as the user table.

You need Control or Alter authority for the user table. The database must be enabled before you enable a table in it.

In the following examples, a table is enabled to hold image data using the default table space. The database is already enabled.

Enabling tables

Using the API: In Figure 17, before enabling the table, the code creates the table and commits changes. The example includes some error-checking code. The complete sample program is in the ENABLE.C file in the SAMPLES subdirectory.

```
/*---- Create a table in the database. -------------------------------*/
printf("%s: Creating table......\n", argv[0]);
sprintf((char*) buffer, (char*) szCreate, tableName, imageColumn);
rc = SQLAllocStmt(hdbc, &hstmt);
cliCheckError(SQL_NULL_HENV, hdbc, SQL_NULL_HSTMT, rc);

/*---- Commit changes to the database. -------------------------------*/
rc = SQLTransact(henv, hdbc, SQL_COMMIT);
cliCheckError(henv, hdbc, SQL_NULL_HSTMT, rc);

/*---- Enable the table. ---------------------------------------------*/
printf("%s: Enabling table......\n", argv[0]);
step="DBiEnableTable";
rc = DBiEnableTable(NULL, "employee");
if (rc < 0) {
        printf("%s: %s failed!\n", argv[0], step);
        printMsg(rc);
        DBiGetError(&sqlcode, errorMsgText);
        printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else if (rc > 0) {
        printf("%s: %s, warning detected.\n", argv[0], step);
        printMsg(rc);
        DBiGetError(&sqlcode, errorMsgText);
        printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else
        printf("%s: %s OK\n", argv[0], step);
```

Figure 17. Sample code that enables a table

Using the db2ext command line: In this example, the table already exists, and the database is enabled.

```
enable table employee for db2image
```

Enabling columns

When you enable a column, the extender adds information to the administrative support tables that belong to the user table. You need Control or Alter authority for the user table the column is in. Both the database and table must be enabled before you enable the column.

In the following examples, the PICTURE column in the EMPLOYEE table is enabled to hold image data. The database and table are already enabled.

Using the API: This example includes some error-checking code. The complete sample program is in the ENABLE.C file in the SAMPLES subdirectory.

```
step="DBiEnableColumn";
rc = DBiEnableColumn("employee", "picture");
if (rc < 0) {
     printf("%s: %s failed!\n", argv[0], step);
     printMsg(rc);
     DBiGetError(&sqlcode, errorMsgText);
     printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else if (rc > 0) {
     printf("%s: %s, warning detected.\n", argv[0], step);
     printMsg(rc);
     DBiGetError(&sqlcode, errorMsgText);
     printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else
     printf("%s: %s OK\n", argv[0], step);
```

Figure 18. Sample code that enables a column

Using the db2ext command line: In this example, the column already exists, and the database and table are enabled.

```
enable column employee picture for db2image
```

Disabling data objects

If you remove extender data from a database, table, or column, you no longer need it to be enabled. You have two ways to disable data objects: the DISABLE commands and the APIs. For more information about the extender commands, see “Chapter 17. Administration Commands for the Client” on page 471. For more information about the extender APIs, see “Chapter 16. Application Programming Interfaces” on page 265.

Before dropping a table or database that contains extender data, disable it and stop the server for that database.

Disabling

Chapter 7. Redistributing Extender Data in a Partitioned Database System (EEE only)

DB2 Extended Enterprise Edition allows you to add and delete database partition servers (also called nodes) in a partitioned database environment. After nodes have been added (or before they are deleted), existing data can be redistributed to take advantage of the new configuration.

Two steps are required to redistribute extender data. First, you must redistribute DB2 data. Then, you can redistribute DB2 extender data.

Redistributing DB2 data

Before you redistribute data, you must redistribute DB2 data using the DB2 command REDISTRIBUTE NODEGROUP.

For more information on redistributing DB2 data, refer to the *DB2 Administration Guide*.

Redistributing extender data

After you have redistributed DB2 data, you are ready to redistribute extender data. Enter the extender command REDISTRIBUTE NODEGROUP to start extender data redistribution.

```
redistribute nodegroup
```

The `REDISTRIBUTE NODEGROUP` command redistributes audio, image, and video extender data, and QBIC feature data, placing it at the same node as its corresponding user data.

If the redistribution process returns an error, you can rerun the command with or without the CONTINUE parameter according to the instructions provided by the command response. This option instructs the system to continue from where it stopped, rather than starting from the beginning. The CONTINUE parameter cannot be used the first time you run the REDISTRIBUTE NODEGROUP command after running DB2's REDISTRIBUTE NODEGROUP command.

To maintain data integrity, redistribute one nodegroup at a time. Wait until one nodegroup has finished redistribution before starting another.

Redistributing data

You must connect to the database before using this command.

You need SYSADM or DBADM authority to run this command.

Chapter 8. Tracking Data Objects and Media Files

As you create and debug applications that use the DB2 extenders, it's useful to know which data objects are enabled for extender data. For example, if you can determine that a certain table is enabled for image data, your application can successfully store image files in that table.

It's also useful to understand the correlation between user tables and external media files. For example, suppose your application creates a report that lists images of employees by department. Each department's data is stored in a separate table that has a PICTURE column for the images. With extenders, you can make sure each table refers to only the images of employees in that department, without having to examine each table individually. You can also discover if your tables refer to files that no longer exist on the system.

Checking the status of data objects

You can check whether databases, tables, and columns are enabled to hold extender data. The following example determines if the current database is enabled for image data. The database is already connected. The complete sample program is in the API.C file in the SAMPLES subdirectory.

Using the API: The sample code in Figure 19 on page 60 includes some error-checking code.

Checking for enablement

```
/*---- Query the database using DBiIsDatabaseEnabled API. ----------*/
step="DBiIsDatabaseEnabled API";
rc = DBiIsDatabaseEnabled(&status);
if (rc < 0) {
     printf("%s: %s FAILED!\n", argv[0], step);
     printMsg(rc);
     DBiGetError(&sqlcode, errorMsgText);
     printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
     fail = TRUE;
} else if (rc > 0) {
     printf("%s: %s, warning detected.\n", argv[0], step);
     printMsg(rc);
     DBiGetError(&sqlcode, errorMsgText);
     printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else {
     if (status == 1) {
         printf("%s: \"%s\" database is enabled for Image Extender\n",
                argv[0], dbName);
         printf("%s: %s PASSED\n\n", argv[0], step);
     } else if (status == 0) {
         printf("%s: \"%s\" database is not enabled for Image Extender\n",
                argv[0], dbName);
         printf("%s: %s PASSED\n\n", argv[0], step);
     } else
         printf("%s: %s FAILED, invalid status!\n", argv[0], step);
}
```

Figure 19. Sample code that checks if a database is enabled

Using the db2ext command line:

```
get extender status
```

Checking the status of user tables and columns is similar to checking the status of a database. Use the DBxIsTableEnabled and DBxIsColumnEnabled APIs, or the GET EXTENDER STATUS command.

Finding table entries that reference files

You can check which entries in user tables refer to an external media file. If you have SYSADM, SYSCTRL, or SYSMAINT authority, you can use the DBxAdminIsFileReferenced API to check which entries in all or a subset of user tables in the current database refer to an external media file. If you have Select authority, you can use the DBxIsFileReferenced API to check which entries in specific user table refer to an external media file.

The following example lists the references to FREDPIC.BMP by the user tables with the schema name ANITAS.

Using the API: The sample code in Figure 20 the following example returns the number of times the file is referenced and where it is referenced. It includes some error-checking code. The complete sample program is in the API.C file in the SAMPLES subdirectory.

```
/*---- Query the database using DBiAdminIsFileReferenced API. ------*/
step="DBiAdminIsFileReferenced API";
rc = DBiAdminIsFileReferenced("anitas", "/usr/lpp/snapshot/fredpic.bmp",
     &count, &filelist);
if (rc < 0) {
     printf("%s: %s FAILED!\n", argv[0], step);
     printMsg(rc);
     DBiGetError(&sqlcode, errorMsgText);
     printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
     fail = TRUE;} else if (rc > 0) {
     printf("%s: %s, warning detected.\n", argv[0], step);
     printMsg(rc);
     DBiGetError(&sqlcode, errorMsgText);
     printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsg Text);
} else {
     if (count == 0)
          printf("%s: \"%s\" file is not referenced\n",
                 argv[0], filename);
     else {          printf("%s: \"%s\" file is referenced %d times\n",
                 argv[0], filename);
          for (i=0; i < count; i++)
             {
                  printf ("filename = %s\n", filelist[i].filename);
                  printf ("     qualifier = %s\n", filelist[i].tqualifier);
                  printf ("     table = %s\n", filelist[i].tname);
                  printf ("     handle = %s\n", filelist[i].handle);
                  printf ("     column = %s\n", filelist[i].column);
                  if (filelist[i].filename)
                          free (filelist[i].filename);
             }
          }
          if (filelist)
                free (filelist);
          printf("%s: %s PASSED\n\n", argv[0], step);
}
```

Figure 20. Sample code that checks if a file is referenced by user tables

Finding files referenced by table entries

If you have SYSADM, SYSCTRL,or SYSMAINT authority, you can use the DBxAdminGetReferencedFiles API or the GET REFERENCED FILES command to list the external media files that are referred to by all or a subset of the user tables in the current database. If you have Select authority, you can

Listing referenced files

use the DBxGetReferencedFiles API or the GET REFERENCED FILES command to list the external media files that are referenced in a specific table.

In the following examples, the files referenced by user tables with the schema name ANITAS are listed.

Using the API: The sample code in Figure 21 returns the number of files it finds and a list of the files. The complete sample program is in the API.C file in the SAMPLES subdirectory.

```
/*---- Query the database using DBiAdminGetReferencedFiles API. ----------*/
step="DBiAdminGetReferencedFilesAPI"
rc = DBiAdminGetReferencedFiles("anitas", &count, &filelist);
if (rc < 0) {
   printf("%s: %s FAILED!\n", argv[0], step);
   printMsg(rc);
   DBiGetError(&sqlcode, errorMsgText);
   printf{"sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
   fail=TRUE;
} else if (rc > 0) {
   printf("%s: %s, warning detected.\n", argv[0], step);
   printMsg(rc);
   DBiGetError(&sqlcode, errorMsgText);
   printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else {
   if (count == 0)
     printf("%s: no referenced files\n", argv[0]);
  else {
     printf("%s: %d referenced files\n", argv[0], count);
     for (i=0; i < count; i++)
       {
   printf ("filename = %s\n", filelist[i].filename);
   printf ("     qualifier = %s\n", filelist[i].tqualifier);
   printf ("     table = %s\n", filelist[i].tname);
   printf ("     handle = %s\n", filelist[i].handle);
   printf ("     column = %s\n", filelist[i].column);
       if (filelist[i].filename)
               free (filelist[i].filename);
    }
   }
   if (filelist)
       free (filelist);
   printf("%s: %s PASSED\n\n", argv[0], step);
}
```

Figure 21. Sample code that gets a list of referenced files

Using the db2ext command line:

```
get referenced files user anitas for db2image
```

Checking if media files exist

Suppose someone deletes a media file from the system but does not update the user table that references it. You might want to list all the inaccessible media files that your user tables reference.

Listing inaccessible files is similar to listing referenced files. If you have SYSADM, SYSCTRL, or SYSMAINT authority, you can use the DBxAdminGetInaccessibleFiles API or the GET INACCESSIBLE FILES command to list the inaccessible media files that are referenced by all or a subset of the user tables in the current database. If you have Select privilege, you can use the DBxGetInaccessibleFiles API or the GET INACCESSIBLE FILES command to list the inaccessible media files that are referenced by a specific table.

Checking for inaccessible media

Chapter 9. Cleaning Up Administrative Support Tables

As you work with the DB2 extenders, obsolete entries can eventually accumulate in the administrative support tables. Someone might delete a media file but not the references to it in the database. Deleting the obsolete metadata can improve performance and reclaim storage space.

Using the API: The sample code in Figure 22 cleans up the image metadata for all user tables owned by ANITAS. It includes some error-checking code. The complete sample program is in the API.C file in the SAMPLES subdirectory.

```
/*---- query database using DBiAdminReorgMetadata API ----*/
step="DBiAdminReorgMetadata API";
rc = DBiAdminReorgMetadata("anitas");
if (rc < 0) {
        printf("%s: %s FAILED!\n", argv[0], step);
        printMsg(rc);
        DBiGetError(&sqlcode, errorMsgText);
        printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
        fail = TRUE;
} else if (rc > 0) {
        printf("%s: %s, warning detected.\n", argv[0], step);
        printMsg(rc);
        DBiGetError(&sqlcode, errorMsgText);
        printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else
        printf("%s: %s PASSED\n\n", argv[0], step);

/*---- end of query using DBiAdminReorgMetadata API ----*/
```

Figure 22. Sample code that cleans up administrative support tables

Using the db2ext command line:

```
reorg database user anitas for db2image
```

If you are not a DBA but have CONTROL authority, you can use the DBxReorgMetadata APIs or the REORG command to clean up metadata for the tables you own.

Part 3. Programming for Image, Audio, and Video Data

Chapter 10. Programming Overview

This chapter provides an overview of programming for the DB2 extenders. It gives information that you need before you begin programming for the extenders, and presents a sample application that illustrates how to code for an extender.

Using extender UDFs and APIs

The DB2 extenders provide user-defined functions to store, access, and manipulate image, audio, and video data in a database. You code requests for these UDFs in your application program using SQL statements in the same way that you request SQL built-in functions. Like built-in functions, UDFs are executed in the database server.

The following SQL statements in a C application program request an Image Extender UDF named DB2Image to store an image in a database table; the content of the source image is in a server file:

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',                          /*id*/
        'Anita Jones',                     /*name*/
        DB2IMAGE(                          /*Image Extender UDF*/
           CURRENT SERVER,                 /*database */
           '/employee/images/ajones.bmp',  /*image content*/
           'ASIS',                         /*keep the image format*/
           :hvStorageType,                 /*store image in DB as BLOB*
           'Anita''s picture')             /*comment*/
       );
```

You use extender application programming interfaces to display images and play audio or video objects. You code these APIs using client function calls in C. The functions are executed in your database client workstation.

The following C statements include an API named DBiBrowse. The API retrieves the data for an image handle and invokes a browser to display the image:

```
EXEC SQL BEGIN DECLARE SECTION;
  char hvImg_hdl[251];
EXEC SQL END DECLARE SECTION
```

```
EXEC SQL SELECT PICTURE INTO :hvImg_hdl
   WHERE NAME= 'Robert Smith';

rc=DBiBrowse(
             "ib %s",                 /*image browser*/
             MMDB_PLAY_HANDLE,        /*use image handle*/
             hvImg_hdl,               /*image handle*/
             MMDB_PLAY_NO_WAIT);      /*run browser independently*/
```

UDFs must run under the user ID of the instance: DB2 extender UDFs must run under the same user ID as the DB2 extender instance. In addition, if you create a DB2 extender instance or use an existing DB2 extender instance, the UDFs must run under the same user ID as the DB2 instance.

DB2 must be configured properly: DB2 must be configured properly to ensure the proper operation of the DB2 extenders, especially the proper operation of DB2 extender UDFs. In particular, the APP_CTL_HEAP_SZ database configuration parameter must be set properly.

Tasks you can perform with extender UDFs and APIs

Table 4 lists the tasks that you can perform with the extender UDFs and APIs and shows where each task is described.

Table 4. Tasks you can perform with DB2 extender APIs

Task	See
Store an image, audio, or video object	Page 83
Retrieve an image, audio, or video object	Page 96
Retrieve and use image, audio, and video attributes	Page 101
Retrieve comments associated with an image, audio, or video object	Page 104
Update an image, audio, or video object	Page 104
Display an image object	Page 119
Display a thumbnail-size image or video frame	Page 122
Play an audio or video object	Page 123
Query images by content	Page 125
Detect video scene changes	Page 167

Sample table for extender examples

Throughout this chapter you will see programming examples that use the DB2 extenders. The examples assume that you created a database table named EMPLOYEE that contains personnel information. The table includes columns for the identification and name of employees. Depending on the extender, the table also includes a column for employee pictures, voice greetings, and video clips.

Figure 23 illustrates the structure of the employee table and shows the SQL statement used to create the table.

```
CREATE TABLE  employee(
    id          CHAR(6),
    name        VARCHAR(40),
    picture     DB2Image,

    sound       DB2Audio,

    video       DB2Video
    );
```

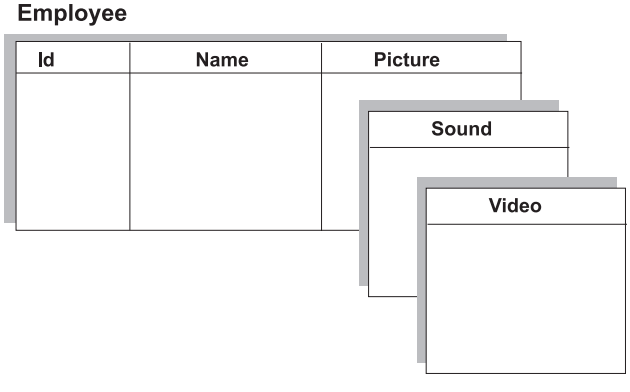


Figure 23. A table used in DB2 extender programming examples

Before you begin programming for DB2 extenders

Before you develop a program that uses the DB2 extenders, you should be familiar with the DB2 application development process and programming techniques as described in *DB2 Embedded SQL Programming Guide*. The process for developing programs that use DB2 extenders is essentially the same as that for traditional DB2 applications.

Your application program code will differ from a traditional DB2 application because of the new data types and functions defined by the extenders. For example, Figure 24 on page 73 shows an application coded in C that uses the Image Extender to identify GIF images stored in a database table. After the images are identified, the program calls an image browser to display them.

As the example illustrates, an application that uses a DB2 extender needs to perform the following functions:

1 Include extender definitions. The `dmbimage.h` file in the example is the include (header) file for the Image Extender. The include file defines the constants, variables, and function prototypes for the extender.

2 Define host variables as necessary to contain input to or output from a UDF, or input to an API call. In the example, `hvFormat`, `hvSize`, `hvWidth`, `hvHeight`, and `hvComment` are host variables that are used to contain data retrieved by the Image Extender UDFs. The host variable `hvImg_hdl` is used to contain an image handle that is specified as input to an Image Extender API call.

3 Specify UDF requests as necessary. In the example, `SIZE`, `WIDTH`, `HEIGHT`, `COMMENT`, and `FORMAT` are Image Extender UDFs.

4 Specify API calls as necessary. In the example, `DBiBrowse` is an API call to a local C function that displays images whose handles are retrieved from a table.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlcodes.h>
#include <dmbimage.h>  1

int count=0;

long
main(int argc,char *argv[])
{
EXEC SQL BEGIN DECLARE SECTION;  2
         char hvImg_hdl[251];                  /* image handle */
         char hvDBName[19];                    /* database name */
         char hvName[40];                      /* employee name */
         char hvFormat[9];                     /* image format */
         long hvSize;                          /* image size */
         long hvWidth;                         /* image width */
         long hvHeight;                        /* image height */
         struct  {
                  short len;
                  char data[32700]
         } hvComment;                          /* comment about the image */
 EXEC SQL END DECLARE SECTION;

/*  Connect to database  */

strcpy(hvDBName, argv[1]);                     /* copy the database name */

EXEC SQL CONNECT TO :hvDBName IN SHARE MODE;
/*
 * Set current function path
*/
EXEC SQL SET CURRENT FUNCTION PATH = mmdbsys, CURRENT FUNCTION PATH;
```

Figure 24. An application that uses a DB2 extender (Part 1 of 2)

Before you begin

```
/*
 * Select (query) using Image Extender UDF
 *
 * The SQL statement below finds all images in GIF format.
 */
EXEC SQL DECLARE c1 CURSOR FOR
         SELECT PICTURE, NAME,                 3
                SIZE(PICTURE), WIDTH(PICTURE),
                HEIGHT(PICTURE), COMMENT(PICTURE)
         FROM EMPLOYEE
         WHERE PICTURE IS NOT NULL AND
               FORMAT(PICTURE) LIKE 'GIF%'
FOR FETCH ONLY;

EXEC SQL OPEN c1;
for (;;) {
         EXEC SQL FETCH c1 INTO :hvImg_hdl, :hvName, :hvSize,
                                :hvWidth, :hvHeight, :hvComment;
         if (SQLCODE != 0)
                   break;

         printf("\nRecord %d:\n", ++count);
         printf("employee name = '%s'\n", hvName);
         printf("image size = %d bytes, width=%d, height=%d\n",
                                                 hvSize, hvWidth, hvHeight);
         hvComment.data[Comment.len]='\0';
         printf("comment len = %d\n", hvComment.len);
 printf("comment = %s\n", hvComment.data);
/*
* The API call below displays the images
*/
 4  rc=DBiBrowse ("ib %s",MMDB_PLAY_HANDLE,hvImg_hdl,
                          MMDB_PLAY_WAIT);
}

EXEC SQL CLOSE c1;

/* end of program */
```

Figure 24. An application that uses a DB2 extender (Part 2 of 2)

Including extender definitions

You need an include (header) file in your application program for each extender that you use. Each include file defines constants, variables, and function prototypes used by the extender. The names of the include files are:

Include file	Extender
dmbimage.h	Image
dmbqbapi.h	Image (query by image content)
dmbaudio.h	Audio

dmbvideo.h	Video
dmbshot.h	Video (scene change detection)

You bring the include file into a C program with the #include directive. For example, the following directive brings in the include file for the Image Extender:

```
#include <dmbimage.h>
```

Specifying UDF and UDT names

The full name of a DB2 Extender UDF is mmdbsys.function-name. The full name of a DB2 extender UDT is mmdbsys.type-name, where mmdbsys is the schema-name of the function or distinct type. For example, the full name of the Content UDF is mmdbsys.Content; the full name of the DB2Image data type created by the Image Extender is mmdbsys.DB2Image. You can omit the mmdbsys schema-name if you previously set the current functionpath to mmdbsys, for example:

```
SET CURRENT FUNCTION PATH = mmdbsys, CURRENT FUNCTION PATH
```

Transmitting large objects

You can transmit large objects such as images, audio clips, and video clips between your application and a DB2 database in various ways. The method you use depends on whether the object is transmitted to or from a file or memory buffer, and whether the file is in your client machine or in the database server machine.

If the object is transmitted between a table and a server file

When you transmit an object between a database table and a server file, specify the file path in the appropriate extender UDF request. Because the extender UDF and the file are both on the server, the extender will be able to find the file. For example, in the following SQL statement, an image whose content is in a server file is stored in a database table:

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2Image(
          CURRENT SERVER,
          '/employee/images/ajones.bmp',
```

```
          'ASIS',
          :hvStorageType,
          'Anita''s picture')
      );
```

If the object is transmitted to or from a client buffer

The extenders cannot directly access a memory buffer. If you want to transmit an object to or from a buffer on your client machine, you need a way to do it other than by specifying a buffer location. One way to transmit an object to or from a buffer is through a host variable. This is the way you normally transmit objects between an application and a DB2 database.

You define and use host variables for large objects in the same way as for traditional character and numeric objects. You declare the host variables in a DECLARE section, assign them values for transmission, or access values transmitted to them.

When you declare a host variable for image, audio, or video data, specify a data type of BLOB. When you use a UDF to store, retrieve, or update an object, you specify the appropriate host variable as an argument in the UDF request. Use the same format as for other host variables that you specify in an SQL statement.

For example, the following SQL statements declare and use a host variable named hvaudio to transmit an audio clip to the database:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB (2M) hvaudio;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2Audio(
          CURRENT SERVER,
          :hvaudio,
          'WAVE',
          CAST(NULL as LONG VARCHAR),
          'Anita''s voice')
      );
```

Using LOB locators

Large objects such as audio and video clips can be very large, and using host variables might not be the most efficient way of manipulating them. A **LOB locator** might be a better way to manipulate LOBs in your applications.

A LOB locator is a small (4-byte) value stored in a host variable that your program can use to refer to a much larger LOB in the DB2 database. Using a

LOB locator, your program can manipulate the LOB as if it was stored in a regular host variable. The difference is that there is no need to transport the LOB between the database server and the application on the client machine. For example, when you select a LOB in a database table, the LOB remains on the server, and the LOB locator moves to the client.

You declare a LOB locator in a DECLARE section and use it in the same way as a host variable. When you declare a LOB locator for image, audio, or video data, specify a data type of BLOB_LOCATOR. For example, the following SQL statements declare and use a LOB locator named `video_loc` to retrieve a video clip from a database table:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_LOCATOR video_loc;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT CONTENT(VIDEO)
     INTO :video_loc
     FROM EMPLOYEE
     WHERE NAME='Anita Jones';
```

UDFs use LOB locators: DB2 extender UDFs that store, retrieve, and update image, audio, and video objects use LOB locators. These UDFs in DB2 extenders V1 did not use LOB locators, and because of this, could not process objects larger than 2 MB. This restriction forced users to transmit objects larger than 2 MB in segments. Because these UDFs now use LOB locators, the 2 MB restriction is removed.

If the object is transmitted to or from a client file

Use a file reference variable to transmit objects to and from a file on a client. Using a file reference variable saves you from having to allocate buffer space for a large object in your application program. When you use a file reference variable with a UDF, DB2 passes the BLOB content directly between the file and the UDF.

You declare a file reference variable in a DECLARE section and use it in the same way as a host variable. When you declare a file reference variable for image, audio, or video data, specify a data type of BLOB_FILE. However, unlike a host variable, which contains the content of an object, the file reference variable contains the name of the file. The size of the file can be no larger than the size of the BLOB defined for the UDF.

You have various options for how to use a file reference variable for input and output. You choose the option you want by setting the FILE_OPTIONS field in the file reference variable structure in your program. You can choose from the following options:

Before you begin

Option for input:

SQL_FILE_READ. This file can be opened, read, and closed. The length of the data in the file (in bytes) is determined when the file is opened. The data_length field of the file reference variable structure holds the length of the file (in bytes).

Options for output:

SQL_FILE_CREATE. This option creates a new file if it doesn't already exist. If the file already exists, an error message is returned. The data_length field of the file reference variable structure holds the length of the file (in bytes).

SQL_FILE_OVERWRITE. This option creates a new file if it does not already exist. If the file already exists, the new data overwrites the data in the file. The data_length field of the file reference variable structure holds the length of the file (in bytes).

SQL_FILE_APPEND. This option appends the output to the file if the file already exists. If the file does not exist, it creates a new file. The data_length field of the file reference variable structure holds the length of the data added to the file (in bytes), not the total length of the file.

For example, the following statements declare a file reference variable named `Img_file` and use it to store an image, whose content is in a client file, into a database table. Notice the SQL_FILE_READ assignment in the FILE_OPTIONS field:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_FILE Img_file;
  EXEC SQL END DECLARE SECTION;

strcpy (Img_file.name,"/employee/images/ajones.bmp");
Img_file.name_length=strlen(Img_file.name);
Img_file.file_options=SQL_FILE_READ;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2Image(
          CURRENT SERVER,
          :Img_file,
          'ASIS',
          CAST(NULL as LONG VARCHAR),
          'Anita''s picture')
       );
```

Specifying file names when you transmit objects

The DB2 extenders give you flexibility in how to specify file names when you store, retrieve, or update objects.

Although you can specify a fully qualified file name, (that is, a complete path followed by the file name) for store, retrieve, and update operations, it's preferable to specify a relative file name. In AIX, HP-UX, and Solaris, a relative file name is any file name that does not begin with a slash; in OS/2 and Windows, a relative file name is any file name that does not begin with a drive letter followed by a colon and backslash.

If you specify a relative file name, the extenders will use the directory specifications in various client and server environment variables as a search path to resolve the file name. A full path name consists of a leading part, which is related to mount points, and a trailing pathname, which uniquely identifies the needed file. The trailing pathname is specified in UDFs. Environment variables supply a list of leading pathnames to search when trying to resolve relative file names. See "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551 for information about the environment variables that the DB2 Extenders use to resolve file names.

The extenders also convert file name formats as appropriate. When a file name is passed to the server, it is converted to the appropriate format for the server's operating system. For example, an OS/2 file name such as c:\dir1\abc.bmp is converted to /dir1/abc.bmp when passed to an AIX server.

Handling return codes

All embedded SQL statements or DB2 CLI calls in your program, including those that invoke DB2 extender UDFs, generate codes that indicate whether the embedded SQL statement or DB2 CLI call executed successfully. Other DB2 extender APIs, such as administrative APIs, also return codes that indicate success or lack of success. Your program should check and respond to the codes returned by embedded SQL statements, CLI calls, and APIs.

For information on handling these return codes, see "Chapter 19. Diagnostic Information" on page 517.

Before you begin

Chapter 11. Storing, Retrieving, and Updating Objects

This chapter describes how to use the DB2 extender user-defined functions to store, retrieve, and update an image, audio, or video.

Image, audio, and video formats

Table 5 lists the formats in which you can store, retrieve, or update image, audio, and video objects. For image objects only, you can have the Image Extender convert the format of the image as it stores, retrieves, or updates it. (Audio and video object formats cannot be converted when stored, retrieved, or updated.)

The Read and Write columns in the table indicate which formats can be read and which formats can be converted when written. When the entry in the Read column in the table is x, the corresponding object format can be used when storing, retrieving, or updating. When the entry in the Write column is x, an object (image only) can be converted to the corresponding format when stored, retrieved, or updated. For example, an image in BMP format can be converted to a GIF format when stored, retrieved, or updated; an image in JPG format can be converted to TIF format; but an image in TIF format cannot be converted to JPG format.

Although listed in the table in uppercase, format specifications in store, retrieve, or update requests are not case sensitive. For example, the specifications GIF, gif, and Gif are equivalent.

Table 5. Formats that can be processed by the DB2 extenders

Format	Description	Read	Write
	Image Formats		
_IM	PS/2 Audio Video Connection (AVC)	x	
BMP	OS/2 - Microsoft Windows bitmap[1]	x	x
EPS	Encapsulated PostScript		x
EP2	Encapsulated level 2 PostScript		x
GIF	Compuserve GIF89a (including animated GIFs[2]) and 87	x	x
IMG	IOCA image	x	x
IPS	Brooktrout FAX card file	x	x
JPG	JPEG[3] (JFIF format)	x	

Formats

Table 5. Formats that can be processed by the DB2 extenders (continued)

Format	Description	Read	Write
PCX	PC paint file (grayscale only)	x	x
PGM	Portable gray map (from PBMPLUS)	x	x
PS	PostScript		x
PSC	Compressed PostScript image		x
PS2	PostScript level 2 (color)		x
TIF	All TIFF 5.0 formats	x	x
YUV	Digital video for YUV	x	x
	Audio formats		
AIF or AIFF	Audio Interchange File Format	x	
AIFFC	Audio Interchange File Format Compressed	x	
AU	Sun audio file format	x	
MIDI	Musical Instrument Digital Interface	x	
MPG1 or MPEG1	Moving Pictures Expert Group 1	x	
WAV or WAVE	Wave	x	
	Video formats		
AVI	Audio/Video Interleaved	x	
MPG1 or MPEG1	Motion Picture Coding Expert Group 1	x	
MPG2 or MPEG2	Motion Picture Coding Expert Group 2	x	
QT	Quicktime (AVI)	x	

Image conversion options

Table 6 on page 83 lists the conversion options (in addition to format conversion) that you can specify for an image when it is stored, retrieved, or updated. The Image Extender applies your specifications to the target image; the source image is not changed.

Each conversion option is specified as a parameter/value pair. The allowed values for each parameter are listed in the table.

1. Read is supported for OS/2 Version 1, OS/2 Version 2, Windows Version 2, Windows Version 3, and Windows NT BMP format. Write is supported for OS/2 Version 1 BMP format.
2. The DB2 Image Extender stores attribute information for only the first image in the animated GIF file.
3. Support uses software that is based in part on the work of the Independent JPEG Group.

Table 6. Image conversion options

Parameter	Description	Value
-b	Number of bits used to represent each image sample	1 or 8 bits
-s[4]	Scaling factor	Any decimal value greater than zero. The scaling factor specifies the size ratio of the converted image to the original. For example, a scaling factor of 0.5 converts the image to half of its original size. A scaling factor of 2.0 converts the image to twice its original size.
-p	Photometric (image inversion). This option applies to black and white or grayscale images only	0 = Ones are black 1 = Ones are white
-r[4]	Rotation	0 = 0 degrees (no rotation) 1 = 90 degrees (counterclockwise) 2 = 90 degrees (clockwise) 3 = 180 degrees
-x[4]	Width in pixels	Number of pixels
-y[4]	Height in pixels	Number of pixels
-c	Compression type	0 = IBM MMR 1 = CCITT Group 3 1-D 2 = CCITT Group 3 2-D (k=2) 3 = CCITT Group 3 2-D (k=4) 4 = CCITT Group 4 6 = TIFF Type 2 10 = Uncompressed 14 = LZW 15 = TIFF Packbits 25 = JBIG

Storing an image, audio, or video object

Use the DB2Image, DB2Audio, or DB2Video UDF in an SQL INSERT statement to store an image, audio, or video object in a database.

You can store an object whose source is in a buffer or file in a client machine or in a server file. For any of these sources, you can store the object in a database table as a BLOB, or in a file on the database server.

When you request the UDF, you need to specify:

4. If you specify this option for an interlaced GIF image, you should also specify a compression type of LZW.

- The name of the currently connected database server; this is contained in the CURRENT SERVER special register.
- The source of the object's content; this is either in a client buffer, client file, or server file.
- Whether you want to store the content in a database table as a BLOB, or on a file server.
- The format of the source.
- A comment to be stored with the object (or a null value or empty string if you don't want to store a comment).

The Image, Audio, and Video Extenders allow you to store an object even if they don't recognize the object's format. In cases where the format is not recognized, you need to specify the attributes of the object. When you store an image or video with user-supplied attributes, you can also store a thumbnail, that is a miniature image representing the image or video.

For images only, you have the option of having the format of the image converted when it is stored. If you request format conversion, you need to specify both the source and target formats of the image. In a format conversion request, you can also specify further changes to the image, such as cropping it or rotating it. You indicate these changes by specifying conversion options.

Commit the store operation: Commit the unit of work after you store an image, audio, or video object in a database, This frees up locks that the extenders hold so that you can perform update operations on the stored object.

DB2Image, DB2Audio, and DB2Video UDF formats

The DB2Image, DB2Audio, and DB2Video UDFs are overloaded, that is, they have different formats depending on how the UDFs are used. Each UDF has the following formats (the xxxxx shown in the formats can be Image, Audio, or Video):

Format 1: Store an object from a client buffer or client file:

```
DB2xxxxx(
     CURRENT SERVER,          /* database name name in CURRENT SERVER REGISTER */
     content,                 /* object content */
     format,                  /* source format */
     target_file,             /* target file name for storage in file server */
                              /* or NULL for storage in table as BLOB */
     comment                  /* user comment */
  );
```

Format 2: Store an object from a server file:

```
DB2xxxxx(
     CURRENT SERVER,          /* database name in CURRENT SERVER REGISTER */
     source_file,             /* source file name */
     format,                  /* source format */
     stortype,                /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                              /* in file server*/
                              /* MMDB_STORAGE_TYPE_INTERNAL=store */
                              /* as a BLOB*/
     comment                  /* user comment */
  );
```

Format 3: Store an object with user-supplied attributes from a client buffer or client file:

```
DB2xxxxx(
     CURRENT SERVER,          /* database name in CURRENT SERVER REGISTER */
     content,                 /* object content */
     target_file,             /* target file name for storage in file server */
                              /* or NULL for storage in table as BLOB */
     comment,                 /* user comment */
     attrs,                   /* user-supplied attributes */
     thumbnail                /* thumbnail (image and video only) */
  );
```

Format 4: Store an object with user-supplied attributes from a server file:

```
DB2xxxxx(
     CURRENT SERVER,          /* database name in CURRENT SERVER REGISTER */
     source_file,             /* source file name */
     stortype,                /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                              /* in file server*/
                              /* MMDB_STORAGE_TYPE_INTERNAL=store */
                              /* as a BLOB*/
     comment,                 /* user comment */
     attrs,                   /* user-supplied attributes */
     thumbnail                /* thumbnail (image and video only) */
  );
```

The DB2Image UDF includes the following additional formats:

Format 5: Store an image from a client buffer or client file with format conversion:

```
DB2Image(
     CURRENT SERVER,          /* database name in CURRENT SERVER REGISTER */
     content,                 /* object content */
     source_format,           /* source format */
     target_format,           /* target format */
     target_file,             /* target file name for storage in file server */
                              /* or NULL for storage in table as BLOB */
     comment                  /* user comment */
  );
```

Format 6: Store an image from a server file with format conversion:

```
DB2Image(
     CURRENT SERVER,           /* database name in CURRENT SERVER REGISTER */
     source_file,              /* server file name */
     source_format,            /* source format */
     target_format,            /* target format */
     target_file,              /* target file name for storage in file server */
                               /* or NULL for storage in table as BLOB */
     comment                   /* user comment */
  );
```

Format 7: Store an image from a client buffer or client file with format conversion and additional changes:

```
DB2Image(
     CURRENT SERVER,           /* database name in CURRENT SERVER REGISTER */
     content,                  /* object content */
     source_format,            /* source format */
     target_format,            /* target format */
     conversion_options,       /* Conversion options */
     target_file,              /* target file name for storage in file server */
                               /* or NULL for storage in table as BLOB */
     comment                   /* user comment */
  );
```

Format 8: Store an image from a server file with format conversion and additional changes:

```
DB2Image(
     CURRENT SERVER,           /* database name in CURRENT SERVER REGISTER */
     source_file,              /* server file name */
     source_format,            /* source format */
     target_format,            /* target format */
     conversion_options        /* conversion options */
     target_file,              /* target file name for storage in file server */
                               /* or NULL for storage in table as BLOB */
     comment                   /* user comment */
  );
```

For example, the following statements in a C application program insert a row that includes an image into the employee table. The source image is in a server file named ajones.bmp. The image is stored in the employee table as a BLOB. (This corresponds to format 2 above.)

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',                         /*id*/
        'Anita Jones',                    /*name*/
        DB2IMAGE(                         /*Image Extender UDF*/
          CURRENT SERVER,                 /*database*/
          '/employee/images/ajones.bmp',  /*source file */
```

```
          'ASIS',                          /*keep the image format*/
          :hvStorageType                   /*store image in DB as BLOB*/
          'Anita''s picture')              /*comment */
      );
```

The following statements in a C application program store the same row into the employee table as in the previous example. However here the image is converted from BMP to GIF format as it is stored. (This corresponds to format 6 above.)

```
EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',                          /*id*/
        'Anita Jones',                     /*name*/
        DB2IMAGE(                          /*Image Extender UDF*
          CURRENT SERVER,                  /*database*/
          '/employee/images/ajones.bmp',   /*source file */
          'ASIS',                          /*source image format*/
          'GIF',                           /*target image format*/
          'Anita''s picture')              /*comment*/
      );
```

When you store an image, audio, or video object, the extender computes attributes such as the number of colors used in the image, audio playing time, or video compression format. If you store an object with an unrecognized format, you need to provide these attributes as input to the UDF. The extender stores the attributes in the database along with other attributes, such as comments about the object and the identification of the user who stored the object. These attributes are then available for you to use in queries.

Storing an object that resides on the client

Use a host variable or a file reference variable to transmit the contents of an image, audio, or video object from a client buffer or client file to the server.

If the object is in a client file, use a file reference variable to transmit its content for storage in the server. For example, the following statements in a C application program define a file reference variable named `Audio_file` and use it to transmit an audio clip whose content is in a client file for storage in a database table on the server. Notice that the `file_option` field of the file reference variable is set to `SQL_FILE_READ` for input. Also notice that the file reference variable is used as the content argument to the DB2Audio UDF.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_FILE Audio_file;
EXEC SQL END DECLARE SECTION;

strcpy (Audio_file.name, "/employee/sounds/ajones.wav");
Audio_file.name_length= strlen(Audio_file.name);
Audio_file.file_options= SQL_FILE_READ;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
```

```
        'Anita Jones',
        DB2AUDIO(
          CURRENT SERVER,
          :Audio_file,          /* file reference variable */
          'WAVE',
          CAST(NULL as LONG VARCHAR),

          'Anita''s voice')
      );
```

If the object is in a client buffer, use a host variable, defined as either BLOB or BLOB_LOCATOR, to transmit its content for storage in the server. In the following C application program statements, a host variable named `Video_loc` is used to transmit the contents of a video clip for storage in the server. The video clip is stored in a database table as a BLOB. Notice that the host variable is used as the content argument to the DB2Video UDF.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_LOCATOR Video_loc;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2VIDEO(
          CURRENT SERVER,
          :Video_loc,         /* host variable */
          'MPEG1',
          '',
          'Anita''s video')
       );
```

Make sure you have enough UDF memory: When you store an object whose content is in a client buffer, you need to make sure that the UDF_MEM_SZ parameter in the Database Manager Configuration is set to 4 MB or greater. You can update the UDF_MEM_SZ parameter by using the DB2 command UPDATE DATABASE MANAGER CONFIGURATION. For more information on the UPDATE DATABASE MANAGER command, see the *DB2 Command Reference*.

Storing an object that resides on the server

When the image, audio, or video you want to store is in a server file, specify its path as the content argument to the UDF. For example, the following statement in a C application program stores a row that includes an image into the database. The image content is in a file on the server. The stored image remains in the server file and is pointed to from the database.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;
```

```
hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2IMAGE(
          CURRENT SERVER,
          '/employee/images/ajones.bmp', /*source in server file */
          'BMP',
          :hvStorageType,
          'Anita''s picture')
       );
```

Specify the correct path: When you store an object whose source is in a server file, you can specify the file's fully qualified name or a relative name. If you specify a relative name, you need to ensure that the appropriate environment variables in the DB2 server include the correct path for the file. For information about setting these environment variables, see "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

Specifying database or file storage

You can store an image, audio, or video object in a database table as a BLOB, or in a server file. If you store the object in a server file, the database points to the file.

If you store the object from a client buffer or client file, you indicate BLOB or server file storage as a result of what you specify in the target_file parameter. If you specify a file name, it indicates that you want to store the object in a server file. If you specify a null value or an empty string, it indicates that you want to store the object as a BLOB in a database table. The data type of the target_file parameter is LONG VARCHAR. If you specify a null value, remember to cast it to a LONG VARCHAR data type.

For example, the following statements in a C application program store a row that includes an image into a database table. The image source is in a client buffer. The image is stored in a server file that the database table points to:

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_LOCATOR Img_buf
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2IMAGE(
          CURRENT SERVER,
          :Img_buf,
```

```
          'ASIS',
          '/employee/images/ajones.bmp',    /* store image in server file */
          'Anita''s picture')
       );
```

If you store an object from a server file, specify the constant MMDB_STORAGE_TYPE_INTERNAL to store the object into a database table as a BLOB. If you want to store the object and have its content remain in the server file, specify the constant MMDB_STORAGE_TYPE_EXTERNAL. MMDB_STORAGE_TYPE_INTERNAL has an integer value of 1. MMDB_STORAGE_TYPE_EXTERNAL has an integer value of 0.

For example, in the following C application program, an audio clip is stored in a server file. The source audio content is already in a server file. The store operation places the filename in the database and thus makes the file accessible through SQL statements.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2AUDIO(
          CURRENT SERVER,
          '/employee/sounds/ajones.wav',
          'WAVE',
          :hvStorageType,             /* store audio in server file */
          'Anita''s voice')
       );
```

Identifying the format for storage

When you store an object, you need to identify its format. The formats that you can specify are listed in Table 5 on page 81. The extenders will store the image, audio, or video object in the same format as the source. For image objects only, you have the option of having the Image Extender convert the format of the stored image. If you choose to have the image format converted, you need to specify the format of the source image and the format of the target image, that is, the image as stored.

Identifying the format for storage without conversion

Specify the format of the source image, audio, or video object when you store the object without format conversion. For example, the following statement in

a C application program stores a bitmap (BMP) image into a database table. The content of the source is in a server file. The target image will have the same format as the source.

```
EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2IMAGE(
          CURRENT SERVER,
          '/employee/images/ajones.bmp',
          'BMP',                          /*image in BMP format */
          '',
          'Anita''s picture')
       );
```

You can also specify a null value or empty string as the format, or for the Image Extender only, the character string ASIS. The extender will then determine the format by examining the source.

Use NULL or ASIS for recognizable formats: Specify a null value, empty string, or ASIS only if the format is recognizable to the extender, that is, if it is one of the formats listed for the extender in Table 5 on page 81. Otherwise, the extender will not be able to store the object.

Identifying the formats and conversion options for storage with format conversion

Specify the format of both the source and target images when you store an image with format conversion. Table 5 on page 81 lists which format conversions are allowed.

In addition, you can specify conversion options that identify additional changes, such as rotation or compression, that you want to apply to the stored image. You specify each conversion option through a parameter and an associated value. The parameters and allowed values are listed in Table 6 on page 83. You can request multiple changes to a stored image by specifying multiple parameter/value pairs.

In the following example, a bitmap (BMP) image, whose content is in a server file, is converted to GIF format when stored in a database table.

```
EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2IMAGE(
          CURRENT SERVER,
          '/employee/images/ajones.bmp',
          'BMP',                          /* source format */
```

```
          'GIF',                          /* target format */
          '',
          'Anita''s picture')
      );
```

In the following example, the image from the previous example is converted to GIF format when stored in a database table. In addition, the image is cropped to a width of 110 pixels and a height of 150 pixels when stored, and it is compressed using LZW compression.

```
EXEC SQL INSERT INTO EMPLOYEE VALUES(
       '128557',
       'Anita Jones',
       DB2IMAGE(
         CURRENT SERVER,
         '/employee/images/ajones.bmp',
         'BMP',                          /* source format */
         'GIF',                          /* target format */
         '-x 110 -y 150 -c 14',          /* conversion options */
         '/employee/images/ajones.gif',
         'Anita''s picture')
      );
```

Storing an object with user-supplied attributes

When you store an image, audio, or video object, you are not limited to formats that the extenders understand. You can specify your own format. Because the extenders do not understand the format, you must specify the attributes of the source object. Assign the attribute values in an attribute structure. The attribute structure must be stored in the data field of the LONG VARCHAR FOR BIT DATA variable in the UDF.

The UDF code on the server always expects data in “big endian format”, a format used by most UNIX platforms. If you are storing an object in “little endian format”, a format typically used in an Intel® and other microprocessor platform, you need to prepare the user-supplied attribute data so that UDF code on the server can correctly process it. (Even if you are not storing the object in little endian format, it is a good idea to prepare the user-supplied attrubute data.) Use the DBiPrepareAttrs API to prepare attributes for image objects, the DBaPrepareAttrs API to prepare attributes for audio objects, and the DBvPrepareAttrs API to prepare attributes for video objects.

For example, the following statements in a C application program store a row that includes an image in a database table. The source image, which is in a server file, has a user-defined format, a height of 640 pixels, and a width of 480 pixels. Notice that the attributes are prepared before the image is stored.

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
struct {
       short len;
```

```
       char data[400];
       }hvImgattrs;
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs.data;
strcpy(pimgattr->format,"FormatI");
pimgattr->width=640;
pimgattr->height=480;
hvImgattrs.len=sizeof(DB2IMAGEATTRS);

DBiPrepareAttrs(pimgattr);

DBEXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2IMAGE(
          CURRENT SERVER,
          '/employee/images/ajones.bmp',
          :hvStorageType,
          'Anita''s picture',
          :hvImgattrs,                  /* user-specified attributes */
          CAST(NULL as LONG VARCHAR)
       );
```

The following statement in a C application program stores a row that includes an audio clip in a database table. The source audio clip, which is in a server file, has a user-defined format, a sampling rate of 44.1 kHz, and has two recorded channels. The audio clip is not MIDI, so empty strings are specified for tracknames and instruments.

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
struct (
       short len;
       char data[600];
}hvAudattr;
EXEC SQL END DECLARE SECTION;

MMDBAudioAttrs          *paudiattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

paudioattr=(MMDBAudioAttrs *) hvAudattr.data;
strcpy(paudioAttr->cFormat,"FormatA");
paudioAttr->ulSamplingRate=44100;
paudioAttr->usNumChannels=2;
hvAudattrs.len=sizeof(MMDBAudioAttrs);

DBaPrepareAttrs(paudioAttr);

EXEC SQL INSERT INTO EMPLOYEE VALUES(
```

```
        '128557',
        'Anita Jones',
        DB2AUDIO(
          CURRENT SERVER,
          '/employee/sounds/ajones.aud',
          :hvStorageType,
          'Anita''s voice',
          :hvAudattr)                  /* user-specified attributes */
       );
```

Storing a thumbnail (image and video only)

When you store an image of your own format, you can also store a **thumbnail**, a miniature-sized version of the image. You control the size and format of the thumbnail. When you store an image in a format that the Image Extender recognizes, it automatically generates and stores a thumbnail for the object. The Image Extender creates a thumbnail in GIF format of size 112 x 84 pixels.

When you store a video object of your own format, you can also store a thumbnail that symbolizes the video object. When you store a video object in a format that the Video Extender recognizes, it automatically stores a generic thumbnail for the object. The Video Extender creates a thumbnail in GIF format of size 108 x 78 pixels.

If you don't want to store a thumbnail when you store an image or video object with user-supplied attributes, specify a null value or empty string in place of the thumbnail.

Generate the thumbnail in your program—the extenders do not provide APIs to generate thumbnails. Create a structure in your program for the thumbnail and specify the thumbnail structure in the UDF.

The following statements in a C application program store a row that includes a video clip in a database table. The source video clip, whose content is in a server file, has a user-defined format. The video content will remain in the server and be pointed to from the table. A thumbnail of a representative video frame is also stored.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
struct {
       short len;
       char data[4000];
       }hvVidattrs;
struct {
       short len;
       char data[10000];
       }hvThumbnail;
EXEC SQL END DECLARE SECTION;
```

```
MMDBVideoAttrs         *pvideoAttr;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

pvideoAttr=(MMDBVideoAttrs *) hvVidattrs.data;
strcpy(pvideoAttr->cFormat,"Formatv");
hvVidattrs.len=sizeof(MMDBVideoAttrs);

/* Generate thumbnail and assign data in video structure */

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2VIDEO(
          CURRENT SERVER,
          '/employee/videos/ajones.vid',
          :hvStorageType,
          'Anita''s video',
          :hvVidattrs,
          :hvThumbnail)                    /* Thumbnail*/
       );
```

Storing a comment

Store a comment with an image, audio, or video object by specifying the comment in the UDF request. A comment is free-form text of data type LONG VARCHAR that can be up to 32,700 bytes long. If you don't want a comment stored when you store an object, specify a null value or empty string in place of the comment. If you specify a null value remember to cast it to a LONG VARCHAR data type.

For example, the following statements in a C application program store a comment with a video clip.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2VIDEO(
          CURRENT SERVER,
          '/employee/videos/ajones.mpg',
          'MPEG1',
          :hvStorageType,
          'Anita''s video')                /* comment */
       );
```

Storing

The following statements in a C application program store an image without a comment.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2IMAGE(
          CURRENT SERVER,
          '/employee/images/ajones.bmp',
          'GIF',
          :hvStorageType,
          Cast(NULL as LONG VARCHAR)            /* no comment */
       );
```

Retrieving an image, audio, or video object

Use the Content UDF in an SQL SELECT statement to retrieve an image, audio, or video object from a database table. You can retrieve the object to a client buffer, client file, or server file.

Content UDF formats for retrieval

The Content UDF is overloaded, meaning, it has different formats depending on how the UDF is used. The formats are as follows:

Format 1: Retrieve an object to a client buffer or client file:

```
Content(
     handle,                    /* object handle */
  );
```

Format 2: Retrieve a segment of an object to a client buffer or client file:

```
Content(
     handle,                    /* object handle */
     offset,                    /* offset where retrieval begins */
     size                       /* number of bytes to retrieve */
  );
```

Format 3: Retrieve an object to a server file:

```
Content(
     handle,                    /* object handle */
     target_file,               /* server file name */
```

```
    overwrite                   /* 0=Do not overwrite target file if it exists */
                                /* 1=Overwrite target file */

  );
```

In addition, the Content UDF includes the following formats for image objects only:

Format 4: Retrieve an image to a client buffer or file with format conversion:

```
Content(
     handle,                    /* object handle */
     target format              /* target format */
       );
```

Format 5: Retrieve an object to a server file with format conversion:

```
Content(
     handle,                    /* object handle */
     target_file,               /* server file name */
     overwrite,                 /* 0=Do not overwrite target file if it exists */
                                /* 1=Overwrite target file */
     target format              /* target format */

  );
```

Format 6: Retrieve an object to a client buffer or file with format conversion and additional changes:

```
Content(
     handle,                    /* object handle */
     target format,             /* target format */
     conversion_options         /* conversion options */
  );
```

Format 7: Retrieve an object to a server file with format conversion and additional changes:

```
Content(
     handle,                    /* object handle */
     target_file,               /* server file name */
     overwrite,                 /* 0=Do not overwrite target file if it exists */
                                /* 1=Overwrite target file */
     target format,             /* target format */
     conversion_options         /* conversion options */
  );
```

For example, the following statement retrieves an image from the employee table to a file on the server. (This corresponds to format 3.)

```
EXEC SQL SELECT CONTENT(                /* retrieval UDF */
       PICTURE,                         /* image handle */
       '/employee/images/ajones.bmp',   /* target file */
       1)                               /* overwrite target file */
FROM EMPLOYEE
WHERE NAME = 'Anita Jones';
```

The following statements in a C application program retrieve an image from the employee table to a file on the server. The format of the image is converted when it is retrieved. (This corresponds to format 5.)

```
EXEC SQL BEGIN DECLARE SECTION;
  char hvImg_fname[255];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT CONTENT(                  /* retrieval UDF */
       PICTURE,                           /* image handle */
       '/employee/images/ajones.bmp',     /* target file */
       1,                                 /* overwrite target file */
       'GIF')                             /* target format */
INTO :hvImg_fname
FROM EMPLOYEE
WHERE NAME = 'Anita Jones';
```

Retrieving an object to the client

You can use the Content UDF to retrieve an image, audio, or video object to a client buffer or client file without format conversion. In addition, you have the option of having the Image Extender convert the format of an image when it is retrieved.

Retrieving an object to a client without format conversion

Use a LOB locator to retrieve an image, audio, or video object to a client buffer without format conversion, or retrieve the LOB. Use a file reference variable to retrieve an image, audio, or video object to a client file.

Retrieving an image, audio, or video object to a client buffer using a host variable, or to a client file using a file reference variable is appropriate when the content of the object is stored in a database table as a BLOB. If the content is in a server file, it might be more efficient to copy the content from the server file to the client file.

Specify the handle of the object. Optionally, you can also specify the offset, starting at byte 1, where retrieval is to start, and the number of bytes that you want to retrieve.

The following statements in a C application program use a LOB locator named `audio_loc` to retrieve an audio clip into a client buffer.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_LOCATOR audio_loc;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT CONTENT(
        SOUND)                                  /* audio handle */
     INTO :audio_loc
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';
```

Make sure you have enough UDF memory: When you retrieve an object to a client buffer, you need to make sure that the UDF_MEM_SZ parameter in the Database Manager Configuration is set to 4MB or greater. You can update the UDF_MEM_SZ parameter with the DB2 command UPDATE DATABASE MANAGER CONFIGURATION. For information on the UPDATE DATABASE MANAGER command, see the publication *DB2 Command Reference*.

Retrieving an image to a client with conversion

Use a LOB locator to retrieve a stored image to a client buffer with format conversion, or retrieve the LOB. Use a file reference variable to retrieve a stored image to a client file with format conversion.

Retrieving an image to a client buffer using a host variable, or to a client file using a file reference variable is appropriate when the content of the image is stored in a database table as a BLOB. If the content is in a server file, it might be more efficient to copy the content from the server file to the client file.

When you retrieve an image with format conversion, you need to specify its target format, that is, the converted format. Table 5 on page 81 identifies the format conversions that are allowed. You can also specify conversion options that identify additional changes, such as rotation or scaling, to be applied to the retrieved image. Table 6 on page 83 lists the conversion options that you can specify.

For example, the following statements in a C application program retrieve an image to a client file. The source image is in bitmap format and is stored in a database table as a BLOB. The retrieved image is converted to GIF and scaled to 3 times its original size.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_FILE Img_file;
EXEC SQL END DECLARE SECTION;

strcpy (Img_file.name, "/employee/images/ajones.gif");
Img_file.name_length= strlen(Img_file.name);
Img_file.file_options= SQL_FILE_CREATE;

EXEC SQL SELECT CONTENT(
        PICTURE,                                /* image handle */
```

```
        'GIF',                              /* target format */
        '-s 3.0')                           /* conversion options */
     INTO :Img_file,
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';
```

Retrieving an object to a server file

You can use the Content UDF to retrieve an image, audio, or video object to a server file without format conversion. In addition, you can use the Content UDF to retrieve an image to a server file with format conversion.

When you retrieve an image, audio, or video object to a file on the server without conversion, specify the object's handle, the target file name, and an overwrite indicator. The overwrite indicator tells the extender whether to overwrite the target file with the retrieved data if the target file already exists on the server. If the target file doesn't exist, the extender creates the target file on the server.

If you specify an overwrite indicator value of 1, the extender overwrites the target file with the retrieved data. If you specify an overwrite indicator value of 0, the extender does not overwrite the target file, thus the data is not retrieved.

The overwrite indicator is ignored if the object to be retrieved is stored in a database table as a BLOB. The target file will be created or overwritten no matter what is specified for the overwrite indicator.

When you retrieve an object to a server file, it returns the name of the server file. For example, the following statement in a C application program retrieves a video to a file on the server. The file name of the server file is stored in the host variable hvVid_fname.

```
EXEC SQL BEGIN DECLARE SECTION;
struct{
       short len;
       char data[250];
      }hvVid_fname[;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT CONTENT(
        VIDEO,                              /* video handle */
        '/employee/videos/ajones.mpg',      /* server file */
        1)                                  /* overwrite target file */
     INTO :hvVid_fname;
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';
```

Using the Content UDF to retrieve an object to a server file without conversion is appropriate when the object is stored in a database table as a

BLOB. If the object is stored in a server file, it might be more efficient to copy the content of the source file to the target file.

When you retrieve an image to a server file with format conversion, specify the image handle, the target file name, an overwrite target indicator, and the target format. Table 5 on page 81 identifies what format conversions are allowed. You can also choose to specify a null value or empty string for the target format or the string ASIS; in this case, the retrieved image will have the same format as the source.

For example, the following statements in a C application program retrieve an image to a file on the server. The source image is in bitmap format and is stored in a database table as a BLOB. The retrieved image is converted to GIF format. The file name of the server file is stored in the host variable hvImg_fname.

```
EXEC SQL BEGIN DECLARE SECTION;
  struct{
        short len;
        char [400];
        }hvImg_fname[;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT CONTENT(
        PICTURE,                              /* image handle */
        '/employee/images/ajones.gif',        /* target file */
        1,                                    /* overwrite target file */
        'GIF')                                /* target format */
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';
```

The server file must be accessible: When you retrieve an object to a server file, you must specify the target file's fully qualified name or ensure that the DB2IMAGEEXPORT, DB2AUDIOEXPORT, and DB2VIDEOEXPORT environment variables are set to properly resolve an incomplete file name specification.

Retrieving and using attributes

When you store an image, audio, or video object in a database, the extender also stores the object's attributes in the database. When you update an object, the extender updates the object's attributes stored in the database. These attributes are available for you to use in queries.

The extenders create UDFs for each of the attributes that they manage. As a result, you can specify UDFs in SQL statements to access and use object attributes. Table 7 on page 102 lists the attributes that the extenders manage and their UDFs. It also indicates the object types for each attribute. Some of

the attributes, such as an object's format and file name, are common to all the object types; that is, they are associated with image, audio, and video objects. Other attributes, such as sampling rate or compression type, are specific to certain object types, such as audio and video.

Table 7. Attributes managed by the DB2 extenders. You can access each attribute through its UDF.

Attribute	UDF	Image	Audio	Video
Name of server file in which the object is stored	Filename	x	x	x
User ID of person who stored the object	Importer	x	x	x
Date and time when the object was stored	ImportTime	x	x	x
Size of the object in bytes	Size	x	x	x
User ID of person who last updated the object	Updater	x	x	x
Date and time when the object was last updated	UpdateTime	x	x	x
Format of the object (for example, GIF or MPEG1)	Format	x	x	x
Comments about the object	Comment	x	x	x
Height of the object (in pixels)	Height	x		x
Width of the object (in pixels)	Width	x		x
Number of colors in the object	NumColors	x		
Thumbnail-size image of the object	Thumbnail	x		x
Number of bytes returned per sample in an audio, or in an audio track of a video	AlignValue		x	x
Number of bits used to represent each sample	BitsPerSample		x	x
Number of recorded channels	NumChannels		x	x
Duration (in seconds)	Duration		x	x
Sampling rate (in samples per second)	SamplingRate		x	x
Average bytes per second transfer time	BytesPerSec		x	
Number of audio track for instrument	FindInstrument		x	
Track number of named track	FindTrackName		x	

Table 7. Attributes managed by the DB2 extenders (continued). You can access each attribute through its UDF.

Attribute	UDF	Image	Audio	Video
Name of recorded instruments	GetInstruments		x	
Track numbers and names of recorded instruments	GetTrackNames		x	
Clock ticks per second of audio	TicksPerSec		x	
Clock ticks per quarter note of audio	TicksPerQNote		x	
Aspect ratio	AspectRatio			x
Video compression format (such as MPEG1)	CompressType			x
Frames per second of throughput	FrameRate			x
Maximum throughput (in bytes per second)	MaxBytesPerSec			x
Number of audio tracks	NumAudioTracks		x	x
Number of frames	NumFrames			x
Number of video tracks	NumVideoTracks			x

You can use an attribute UDF in an SQL statement SELECT clause expression or WHERE clause search condition. When you request the UDF, you specify the name of the column in the database table that contains the object's handle.

For example, the following statement uses the Updater UDF in the SELECT clause of an SQL SELECT statement to retrieve the user ID of the person who last updated an image in the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvUpdatr[30];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT UPDATER(PICTURE)
     INTO :hvUpdatr
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';
```

The following statement uses the Filename UDF in the SELECT clause of a SELECT statement and the NumAudioTracks UDF in the WHERE clause to find videos stored in the employee table that have audio tracks:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvVid_fname[251];
EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL SELECT FILENAME(VIDEO)
     INTO :hvVid_fname
     FROM EMPLOYEE
     WHERE NUMAUDIOTRACKS(VIDEO)>0;
```

Retrieving comments

Use the Comment UDF to retrieve comments stored with an image, audio, or video object. When you retrieve a comment for an object, you specify the column in the database table that contains the object's handle. For example, the following statement retrieves a comment stored with an audio clip in the employee table.

```
EXEC SQL BEGIN DECLARE SECTION;
struct {
       short len;
       char data[32700];
       }hvComment
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT COMMENT(SOUND)
     INTO :hvComment
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';
```

You can also use the Comment UDF as a predicate in the WHERE clause of an SQL query. For example, the following statement retrieves the file name of all images in the employee table that have been noted as "touched up".

```
EXEC SQL BEGIN DECLARE SECTION;
struct {
       short len;
       char data[250];
       }hvImg_fname
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE COMMENT(PICTURE)
           LIKE '%touch%up';
```

Updating an image, audio, or video object

Use the Content UDF in an SQL UPDATE statement to update an image, audio, or video object in a database table. Use the Replace UDF in an SQL UPDATE statement to update an image, audio, or video in a database table and update a comment associated with the object. In either case, the extender updates the attributes associated with the object.

You can update an object that is stored in a database table as a BLOB or stored in a server file (and pointed to from the database). The source of the update can be in a buffer, client file, or server file.

Table 5 on page 81 lists the formats in which you can update image, audio, and video objects. However, you can also update an object whose format is unrecognized by the extender. In this case the object's attributes were specified by the user when the object was stored. When you update an object with user-specified attributes, you need to specify the updated attributes of the object.Use the ContentA UDF in an SQL UPDATE statement to update an image, audio, or video object with user-supplied attributes in a database. Use the ReplaceA UDF in an SQL UPDATE statement to update an image, audio, or video with user-supplied attributes in a database table and update a comment associated with the object. When you update an object with user-specified attributes, you need to specify the attributes of the object, its format, and for video objects only, its compression format.

You can also update the thumbnail for a stored image or video.

Commit the update operation: Commit the unit of work after you update an image, audio, or video object in a database, This frees up locks that the extenders hold so that you can perform subsequent update operations on the stored object.

Content UDF formats for updating

The Content UDF is overloaded, meaning, it has different formats depending on how the UDF is used. The formats are as follows:

Format 1: Update an object from a client buffer or client file:

```
Content(
     handle,                    /* object handle */
     content,                   /* object content */
     source_format,             /* source format */
     target_file                /* target file name for storage in file */
                                /* server or NULL for storage in table as BLOB */
  );
```

Format 2: Update an object from a server file:

```
Content(
     handle,                    /* object handle */
     source_file,               /* server file name */
     source_format,             /* source format */
     stortype                   /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                                /* in file server */
                                /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB*/
  );
```

Updating

Format 3: Update an object with user-supplied attributes from a client buffer or client file:

```
Content(
     handle,                  /* object handle */
     content,                 /* object content */
     target_file,             /* target file name for storage in file server */
                              /* or NULL for storage in table as BLOB */
     attrs,                   /* user-supplied attributes */
     thumbnail                /* thumbnail (image and video only) */
  );
```

Format 4: Update an object with user-supplied attributes from a server file:

```
Content(
     handle,                  /* object handle */
     source_file,             /* source file name */
     stortype,                /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                              /* in file server*/
                              /* MMDB_STORAGE_TYPE_INTERNAL=store */
                              /* as a BLOB*/
     attrs,                   /* user-supplied attributes */
     thumbnail                /* thumbnail (image and video only) */
  );
```

For image objects only, the Content UDF has the following additional formats:

Format 5: Update an image from a client buffer or client file with format conversion:

```
Content(
     handle,                    /* object handle */
     content,                   /* object content */
     source format,             /* source format */
     target format,             /* target format */
     target_file                /* target file name for storage in file server */
                                /* or NULL for storage in table as BLOB */
  );
```

Format 6: Update an object from a server file with format conversion:

```
Content(
     handle,                    /* object handle */
     source_file,               /* server file name */
     source format,             /* source format */
     target format,             /* target format */
     target_file                /* target file name for storage in file server */
                                /* or NULL for storage in table as BLOB */
  );
```

Format 7: Update an image from a client buffer or client file with format conversion and additional changes:

```
Content(
     handle,                    /* object handle */
     content,                   /* object content */
     source format,             /* source format */
     target format,             /* target format */
     conversion_options,        /* conversion options */
     target_file                /* target file name for storage in file server */
                                /* or NULL for storage in table as BLOB */
  );
```

Format 8: Update an object from a server file with format conversion and additional changes:

```
Content(
     handle,                    /* object handle */
     source_file,               /* server file name */
     source format,             /* source format */
     target format,             /* target format */
     conversion_options,        /* conversion options */
     target_file                /* target file name for storage in file server */
                                /* or NULL for storage in table as BLOB */
  );
```

For example, the following statements in a C application program update an image in the employee table. The source content for the update is in a server file named ajones.bmp. The updated image is stored in the employee table as a BLOB. (This corresponds to format 2 above.)

```
EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=CONTENT(
                PICTURE,                        /*image handle*/
                '/employee/newimg/ajones.bmp',  /*source file */
                'ASIS',                         /*keep the image format*/
                '');                            /*store image in DB as BLOB*/
   WHERE NAME='Anita Jones';
```

The following statements in a C application program update the same image as in the previous example. However, here the image is converted from BMP to GIF format on update. (This corresponds to format 6 above.)

```
EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=CONTENT(
                PICTURE,                        /*image handle*/
                '/employee/newimg/ajones.bmp',  /*source file */
                'BMP',                          /*source format*/
                'GIF',                          /*target format*/
                '');                            /*store image in DB as BLOB*/
   WHERE NAME='Anita Jones';
```

Replace UDF formats for updating

The Replace UDF is overloaded, that is, it has different formats depending on how the UDF is used. The formats are as follows:

Updating

Format 1: Update an object from a client buffer or client file and update its comment:

```
Replace(
     handle,                    /* object handle */
     content,                   /* object content */
     source_format,             /* source format */
     target_file,               /* target file name for storage in file */
     comment                    /* user comment */
  );
```

Format 2: Update an object from a server file and update its comment:

```
Replace(
     handle,                    /* object handle */
     source_file,               /* server file name */
     source_format,             /* source format */
     stortype,                  /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                                /* in file server*/
                                /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB*/
     comment                    /* user comment */
  );
```

Format 3: Replace an object with user-supplied attributes from a client buffer or client file and update its comment:

```
Replace(
     handle,                    /* object handle */
     content,                   /* object content */
     target_file,               /* target file name for storage in file */
                                /* or NULL for storage in table as BLOB */
     comment,                   /* user comment */
     attrs,                     /* user-supplied attributes */
     thumbnail                  /* thumbnail */
  );
```

Format 4: Store an object with user-supplied attributes from a server file:

```
Replace(
     handle,                    /* object handle */
     source_file,               /* server file name */
     stortype,                  /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                                /* in file server*/
                                /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB*/
     comment,                   /* user comment */
     attrs,                     /* user-supplied attributes */
     thumbnail                  /* thumbnail */
  );
```

For image objects only, the Replace UDF has the following additional formats:

Format 5: Update an image from a client buffer or client file with format conversion and update its comment:

```
Replace(
     handle,                    /* object handle */
     content,                   /* object content */
     source_format,             /* source format */
     target_format,             /* target format */
     target_file,               /* target file name for storage in file server */
                                /* or NULL for storage in table as BLOB */
     comment                    /* user comment */
  );
```

Format 6: Update an object from a server file with format conversion and update its comment:

```
Replace(
     handle,                    /* object handle */
     source_file,               /* server file name */
     source_format,             /* source format */
     target_format,             /* target format */
     target_file,               /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                                /* in file server */
                                /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB*/
     comment                    /* user comment */
  );
```

Format 7: Update an image from a client buffer or client file with format conversion and additional changes and update its comment:

```
Replace(
     handle,                    /* object handle */
     content,                   /* object content */
     source_format,             /* source format */
     target_format,             /* target format */
     conversion_options,        /* conversion options */
     target_file,               /* target file name for storage in file server */
                                /* or NULL for storage in table as BLOB */
     comment                    /* user comment */
  );
```

Format 8: Update an object from a server file with format conversion and additional changes and update its comment:

```
Replace(
     handle,                    /* object handle */
     source_file,               /* server file name */
     source_format,             /* source format */
     target_format,             /* target format */
     conversion_options,        /* conversion options */
     target_file,               /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                                /* in file server */
                                /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB*/
     comment                    /* user comment */
  );
```

For example, the following statements in a C application program update an audio clip in the employee table and update its associated comment. The

source content for the update is in a server file named ajones.wav. The updated audio clip is stored in the employee table as a BLOB without format conversion (the Audio Extender does not support format conversion). This corresponds to format 2 above.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

EXEC SQL UPDATE EMPLOYEE
   SET SOUND=REPLACE(
                SOUND,                          /*audio handle*/
                '/employee/newaud/ajones.wav', /*source file */
                'WAV',                          /*keep the audio format*/
                :hvStorageType,                 /*store audio in DB as BLOB*/
                'Anita''s new greeting')        /*user comment*/
   WHERE NAME= 'Anita Jones';
```

In the following example an image and its associated comment are updated. The source content for the update is in a server file. The updated image is stored in the employee table as a BLOB, and is converted from BMP to GIF format on update. (This corresponds to format 6 above.)

```
EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=REPLACE(
                PICTURE,                          /*image handle*/
                '/employee/newimg/ajones.bmp',    /*source file */
                'BMP',                            /*source format*/
                'GIF',                            /*target format*/
                ''                                /*store image in DB as BLOB*/
                'Anita''s new picture')
   WHERE NAME='Anita Jones';                      /* user comment */
```

Updating an object from the client

Use a host variable or a file reference variable to update an image, audio, or video object from a client buffer or client file.

If the source for the update is in a client file, use a file reference variable to transmit its content. For example, the following statements in a C application program define a file reference variable named `Audio_file` and use it to update an audio clip stored in a database table as a BLOB. The source for the update is in a client file. Notice that the `file_options` field of the file reference variable is set to `SQL_FILE_READ`, that is, for input. Also notice that the file reference variable is used as the content argument to the Content UDF.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB_FILE Audio_file;
EXEC SQL END DECLARE SECTION;

strcpy (Audio_file.name, "/employee/newsound/ajones.wav");
```

```
Audio_file.name_length= strlen(Audio_file.name);
Audio_file.file_options= SQL_FILE_READ;

EXEC SQL UPDATE EMPLOYEE
   SET SOUND=CONTENT(
                SOUND,
                :Audio_file                /*file reference variable*/
                'WAVE',                    /*keep the image format*/
                CAST(NULL as LONG VARCHAR))

   WHERE NAME='Anita Jones';
```

If the object is in a client buffer, use a host variable to transmit its content for update. In the following C application program example, a host variable named Video_seg is used to transmit the contents of a video clip for update. The comment associated with the video clip is also updated. The video clip is stored in a database table as a BLOB. Notice that the host variable is used as the content argument to the Replace UDF.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB (2M) Video_seg
EXEC SQL END DECLARE SECTION;

EXEC SQL UPDATE EMPLOYEE
   SET VIDEO=REPLACE(
                VIDEO,
                :Video_seg                 /*host variable*/
                'MPEG1',
                CAST(NULL as LONG VARCHAR),

                'Anita''s new video')
   WHERE NAME='Anita Jones';
```

Make sure you have enough UDF memory: When you update an object whose content is in a client buffer, you need to make sure that the UDF_MEM_SZ parameter in the Database Manager Configuration is set to 4MB or greater. You can update the UDF_MEM_SZ parameter with the DB2 command UPDATE DATABASE MANAGER CONFIGURATION.

Updating an object from the server

When the source content for an image, audio, or video object update is in a server file, specify the file path as the content argument to the UDF. For example, the following statement in a C application program updates an image in a database. The image content is in a server file pointed to from the database. The source for the update is also in a server file.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;
```

```
EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=CONTENT(
                PICTURE,                                /* image handle */
                '/employee/newimg/ajones.bmp',          /* source file */
                'ASIS',
                :hvStorageType)
   WHERE NAME='Anita Jones';
```

Specify the correct path: When you update an object whose source is in a server file, you can specify the file's fully qualified name or a relative name. If you specify a relative name, you need to ensure that the appropriate environment variables in the DB2 server includes the correct path for the file. For information about setting these environment variables, see "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

Specifying database or file storage for updates

You can update an image, audio, or video object that is stored in a database table as a BLOB, or in a server file (and pointed to from the database).

If you update an object from a client buffer or client file, you indicate BLOB or server file storage as a result of what you specify in the filename parameter. If you specify a file name, it indicates that you want to update an object whose content is in a server file. If you specify a null file name, it indicates that you want to update an object that is stored as a BLOB in a database table.

For example, the following statements in a C application program update an image whose content is in a server file. The update source is in a client buffer. The image comment is updated, too.

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS BLOB (2M) Img_buf
EXEC SQL END DECLARE SECTION;

EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=REPLACE(
                  PICTURE,
                  :Img_buf,
                  'ASIS',
                  '/employee/newimg/ajones.bmp',      /*update image in*/
                                                      /*server file*/
                  'Anita''s new picture')
   WHERE NAME='Anita Jones';
```

If you update an object from a server file, specify MMDB_STORAGE_TYPE_INTERNAL to update an object stored in a database table as a BLOB. If you want to update an object whose content is in the server file, specify MMDB_STORAGE_TYPE_EXTERNAL.

For example, in the following C application program, an audio clip is updated. The content of the audio clip is in a server file. The source for the update is also in a server file.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

EXEC SQL UPDATE EMPLOYEE
   SET SOUND=CONTENT(
                SOUND,
                '/employee/newimg/ajones.wav',
                'WAVE',
                :hvStorageType)        /*update audio in server file*/
   WHERE NAME='Anita Jones';
```

Identifying the format for update

When you update an object, you need to identify its format. The extenders will store the updating image, audio, or video object in the same format as the source. For image objects only, you have the option of having the Image Extender convert the format of the updated image. If you want to have the image format converted, you need to specify the format of the update source and the format of the target image, that is, the updated image as stored.

Identifying the format for update without conversion

Specify the format of the source image, audio, or video object when you update an object without format conversion. For example, the following statement in a C application program updates a bitmap (BMP) image whose content is in a server file. The format of the updated image will not be converted.

```
EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=CONTENT(
                PICTURE,
                '/employee/newimg/ajones.bmp',
                'BMP',                          /*image format*/
                '')
   WHERE NAME='Anita Jones';
```

You can also specify a null value or empty string as the format, or for the Image Extender only, the character string ASIS. The extender will then determine the format by examining the source.

Use NULL or ASIS for recognizable formats: Specify a null value, empty string, or ASIS only if the format is recognizable to the extender, that is, if it is one of the formats listed for the extender in Table 5 on page 81. Otherwise, the extender cannot update the object.

Identifying the formats and conversion options for update with format conversion

Specify the format of both the source and target images when you update an image with format conversion. Table 5 on page 81 lists which format conversions are allowed.

In addition, you can specify conversion options that identify additional changes, such as rotation or compression, that you want to apply to the updated image. You specify each conversion option through a parameter and an associated value. The parameters and allowed values are listed in Table 6 on page 83. You can request multiple changes to the updated image by specifying multiple parameter/value pairs.

In the following example, an image whose content is in a server file is updated. The source of the update is in bitmap (BMP) format. The format will be converted from BMP to GIF on update.

```
EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=CONTENT(
                PICTURE,
                '/employee/newimg/ajones.bmp',
                'BMP',                              /*source format*/
                'GIF',                              /*target format*/
                '')
   WHERE NAME='Anita Jones';
```

In the following example, the same image is converted to GIF format when updated. In addition, the image is rotated 90 degrees clockwise when updated.

```
EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=CONTENT(
                PICTURE,
                '/employee/newimg/ajones.bmp',
                'BMP',                              /*source format*/
                'GIF',                              /*target format*/
                '-r 1',                             /* conversion options */
                '')
   WHERE NAME='Anita Jones';
```

Updating an object with user-supplied attributes

When you update an image, audio, or video object that was stored with user-supplied attributes, you must specify the attributes of the updating content. Assign the attribute values in an attribute structure. The attribute structure must be stored in the data field of the LONG VARCHAR FOR BIT DATA variable in the UDF.

The UDF code on the server always expects data in “big endian format”, a format used by most UNIX platforms. If you are storing an object in “little endian format”, a format typically used in an Intel and other microprocessor platform, you need to prepare the user-supplied attribute data so that UDF code on the server can correctly process it. (Even if you are not storing the object in little endian format, it is a good idea to prepare the user-supplied attrubute data.) Use the DBiPrepareAttrs API to prepare attributes for image objects, the DBaPrepareAttrs API to prepare attributes for audio objects, and the DBvPrepareAttrs API to prepare attributes for video objects.

For example, the following statements in a C application program update an image whose content is in a server file. The image has a user-defined format, a height of 640 pixels, and a width of 480 pixels. Notice that the attributes are prepared before the image is updated.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
struct {
       short len;
       char data[400];
       }hvImgattrs;
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs.data;
strcpy(pimgattr->Format,"FormatI");
pimgattr->width=640;
pimgattr->height=480;
hvImgattrs.len=sizeof(DB2IMAGEATTRS);

DBiPrepareAttrs(pimgattr);

EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=REPLACE(
                PICTURE,
                '/employee/newimg/ajones.bmp',
                :hvStorageType,
                'Anita''s new picture',
                :ImgAttrs,                      /*user-supplied attributes*/
                CAST(NULL as LONG VARCHAR))
   WHERE NAME='Anita Jones';
```

Updating a thumbnail (image and video only)

Use the Thumbnail UDF to update a thumbnail stored for an image or video object (or add a thumbnail if none is associated with the stored image or video). When you use the Thumbnail UDF, specify the handle of the object whose thumbnail is being updated, and specify the content of the updated (or new) thumbnail.

Updating

Generate the thumbnail in your program—the extenders do not provide APIs to generate thumbnails. You control the size and format of the updating thumbnail. Create a structure in your program for the thumbnail, and specify the thumbnail structure in the UDF.

For example, the following statements in a C application program update the thumbnail associated with a stored video clip.

```
EXEC SQL BEGIN DECLARE SECTION;
     struct {
       short len;
       char data[10000];
     }hvThumbnail;
EXEC SQL END DECLARE SECTION;

/*Create thumbnail and store in hvThumbnail*/

EXEC SQL UPDATE employee
   SET picture=Thumbnail(
                picture,
                :hvThumbnail)
   WHERE name='Anita Jones';
```

You can also update a thumbnail when you update an image or video object with user-supplied attributes. In fact, if you update an image or video with user-supplied attributes, you must specify a thumbnail as input. If you don't want to update the thumbnail when you update the object, specify a null value or empty string in place of the thumbnail specification.

The following statements in a C application program update a video clip with user-supplied attributes, and update a thumbnail associated with the video.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
   struct {
       short len;
       char data[400];
   }hvVidattrs;
   struct {
       short len;
       char data[10000];
   }hvThumbnail;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

MMDBVideoAttrs         *pvideoAttr;
pvideoAttr=(MMDBVideoAttrs *)hvVidattrs.data;
strcpy(pvideoAttr->cformat,"Formatv");
hvVidattrs.len=sizeof(MMDBVideoAttrs);

/* Update video content and thumbnail */
```

```
EXEC SQL UPDATE EMPLOYEE
   SET VIDEO=REPLACE(
                VIDEO,
                '/employee/newvid/ajones.mpg',
                :hvStorageType,
                'Anita''s new video',
                :VidAttrs,
                :hvThumbnail)              /*thumbnail*/
   WHERE NAME='Anita Jones';
```

Updating a comment

You can update a comment by itself, or you can update a comment when you update its associated object.

Use the Comment UDF to update a comment by itself. Specify the content of the updated comment as well as the table column that contains the object's handle. Use a host variable to transmit the content to the server. For example, the following statements declare a host variable named hvRemarks and use it to update an existing comment for a stored video clip.

```
EXEC SQL BEGIN DECLARE SECTION;
  struct {
         short len;
         char data [40];
  }hvRemarks;
EXEC SQL END DECLARE SECTION;

/* Get the old comment */

EXEC SQL SELECT COMMENT(VIDEO)
     INTO :hvRemarks
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';

/* Append to old comment */

hvRemarks.data[Remarks.len]='\0';
hvRemarks.len=strlen(hvRemarks.data);
strcat (hvRemarks.data, "Updated video");
EXEC SQL UPDATE EMPLOYEE
     SET VIDEO=COMMENT(VIDEO, :hvRemarks)
     WHERE NAME = 'Anita Jones';
```

Use the Replace UDF to update a comment when you update its associated object. For example, the following statements update a video clip stored in a server file as well as its associated comment.

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;
```

Updating

```
EXEC SQL UPDATE EMPLOYEE
   SET VIDEO=REPLACE(
                VIDEO,
                '/employee/newvid/ajones.mpg',
                'MPEG1',
                :hvStorageType,
                'Anita''s new video')        /*updated comment*/
   WHERE NAME='Anita Jones';
```

Chapter 12. Displaying or Playing an Image, Audio, or Video Object

This chapter describes how to use the DB2 Extender application programming interfaces to display or play an image, audio or video object stored in a database.

Using the display or play APIs

You can use extender APIs to display an image or video frame stored in a database. You can display a thumbnail-size version or full-size version of an image or video frame. You can also use extender APIs to play audio or video objects stored in a database.

Use the following APIs to display or play objects:

Use this API	To
DBiBrowse	Display an image or video frame
DBaPlay	Play an audio clip
DBvPlay	Play a video clip or display a video frame

When you request any of these APIs, you need to specify:

- The name of the display or play program
- Whether the object to be displayed or played is stored in a database table as a BLOB, or is in a file pointed to from the table
- The name of the source file, or the handle stored in the database table
- Whether you want your application program to wait for the user to close the display or play program before proceeding

Identifying a display or play program

Specify the name of the image browser, audio player, or video player you want to use. Follow the name with %s. The extender will replace the %s with the file that holds the object content. For example, the following statement in a C application program starts the OS/2 image browser (ib) to display an image:

```
rc = DBiBrowse(
     "ib %s",                  /* image display program */
     MMDB_PLAY_FILE,
     "/employee/images/ajones.bmp",
     MMDB_PLAY_NO_WAIT
     );
```

You can also specify a null value instead of naming a specific display or play program. In this case, the extender starts the default image browser, audio player, or video player named in the DB2IMAGEBROWSER, DB2AUDIOPLAYER, or DB2VIDEOPLAYER environment variables. For more information about how the DB2 Extenders use environment variables, see "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

For example, the following statement in a C application program starts the default audio player identified in the DB2AUDIOPLAYER environment variable:

```
rc = DBaPlay(
     NULL,                              /* use default audio player */
     MMDB_PLAY_FILE,
     "/employee/sounds/ajones.wav",
     MMDB_PLAY_NO_WAIT
     );
```

The environment variable must name a program: If you request a default display or play program (by specifying a null value), ensure that the appropriate environment variable specifies a display or play program. If a program is not specified, the API will return an error code.

Specifying BLOB or file content

You can display or play an object stored in a database table as a BLOB or whose content is stored in a file (and pointed to from the database table). If the object is stored as a BLOB, specify MMDB_PLAY_HANDLE. If the object content is stored in a file, specify MMDB_PLAY_FILE. MMDB_PLAY_HANDLE and MMDB_PLAY_FILE are constants defined by the extenders.

For example, the following statement in a C application program plays a video whose content is in a file:

```
rc = DBvPlay(
     "explore %s",
     MMDB_PLAY_FILE,                    /* content in file */
     "/employee/videos/ajones.mpg",
     MMDB_PLAY_NO_WAIT
     );
```

Display and play programs typically accept input from a file. If you specify MMDB_PLAY_FILE, the extender will resolve the file's relative file name, and resolve its path using the value in environment variables. The extender then starts the browse program and passes it the file name. If you specify MMDB_PLAY_HANDLE, the extender extracts the file name from the handle (provided that the file name is not null). If the file name in the handle is null, the object is stored as a BLOB; the extender will create a temporary file in the

client and copy the content of the object from the database table to the client file. The extender will then start the program and pass it the name of the file (or temporary file) that holds the content.

For example, the following statements in a C application program get the handle of an image stored as a BLOB and use the handle to display the image:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvImg_hdl[251];
EXEC SQL END DECLARE SECTION;

rc = DBiBrowse(
     "ib %s",
     MMDB_PLAY_HANDLE,                  /* content is BLOB */
     hvImg_hdl,
     MMDB_PLAY_NO_WAIT
     );
```

The content must be accessible: Make sure that the display or play program can access the object content. If the content is in a server file, but the program requires the content on the client, copy the file to a client file or use the Content UDF. If the content is stored as a BLOB, the extender will automatically retrieve it to the client.

Specifying a wait indicator

You can specify whether you want your application program to wait for the user to end the display or play program before the application continues (that is, before the DBiBrowse, DBaPlay, or DBvPlay API returns a code). If you want your application program to wait, specify MMDB_PLAY_WAIT. If you do not want your application program to wait, specify MMDB_PLAY_NO_WAIT. MMDB_PLAY_WAIT and MMDB_PLAY_NO_WAIT are constants defined by the extenders.

If you specify MMDB_PLAY_WAIT, the display or play program will run in the same thread or process as your application program. If you specify MMDB_PLAY_NO_WAIT, the display or play program will run in its own thread or process independently of your application program.

For example, as a result of the following statement, the application program will wait for the user to close the image browser before the application continues:

```
rc = DBiBrowse(
     "explore %s",
     MMDB_PLAY_FILE,
     "/employee/images/ajones.bmp",
     MMDB_PLAY_WAIT                  /* wait for browser to close */
     );
```

Be careful if you specify DBxPlay and MMDB_PLAY_NO_WAIT: When you issue DBaPlay or DBvPlay, the extender will create a temporary file if the object is stored as a BLOB, if the relative filename cannot be resolved using the values in environment variables, or if the file is not accessible on the client machine. The temporary file is created in the directory specified by the TMP environment variable. If you specify MMDB_PLAY_WAIT, the extender deletes the temporary file after the object is played. However, if you specify MMDB_PLAY_NO_WAIT, the temporary file is not deleted. You will have to delete the temporary file yourself.

Displaying a thumbnail-size image or video frame

A thumbnail is a miniature version of a stored image or video frame. When you store an image in the database, the Image Extender stores a thumbnail of the image in an attribute table. When you store a video in the database, the Video Extender stores a thumbnail of a selected video frame in an attribute table; if the format of the video is recognized by the Video Extender, it stores a generic thumbnail that symbolizes the video object.

By default, the size of an image thumbnail automatically created by the Image Extender is approximately 112 x 84 pixels, and the size of the generic video thumbnail that the Video Extender inserts is 108 x78 pixels; both are stored in GIF format. Depending on the density of data in the image or video frame, this corresponds to approximately 4.5 KB to 5 KB of data. If you store or update an image or video with user-supplied attributes, you can specify a thumbnail of a size and format that you choose.

Use the Thumbnail UDF in an SQL SELECT statement to retrieve a thumbnail from the database, and use a file reference variable to transmit the thumbnail to a file. When you specify the UDF, you need to specify the name of the column in the database table that contains the image or video handle. Then use the DBiBrowse API to display the image or video frame thumbnail.

For example, the following statements retrieve a thumbnail image and then display it:

```
long rc, outCount;
char Thumbnail_filename[254];
FILE *file_handle;

EXEC SQL BEGIN DECLARE SECTION;
  struct {
         short len
         char data[10000];
         }Thumbnail_buffer;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT THUMBNAIL(PICTURE)
```

```
     INTO :Thumbnail_buffer
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';

strcpy (Thumbnail_filename,"/tmp/ajones.tmb");
file_handle=fopen(Thumbnail_filename,"wb+");
outCount=fwrite(Thumbnail_buffer.data, 1, Thumbnail_buffer.len, file_handle);
fclose(file_handle);
rc = DBiBrowse (
               NULL,                        /* use the default display program */
               MMDB_PLAY_FILE,              /* thumbnail image in file */
               Thumbnail_filename,          /* thumbnail image content */
               MMDB_PLAY_WAIT);             /* wait for user to finish */
```

Displaying a full-size image or video frame

Use the DBiBrowse API to display an image stored in a database table. See “Using the display or play APIs” on page 119 for detailed information on using this API.

Use the DBvGetNextFrame API or DBvSeekFrame API to get a full-size video frame. The frame is stored in YUV format in a buffer, and can be converted to RGB format using the DBvFrameDatato24BitRGB API. Append a header to the converted frame (for example, append a BMP file-type header) and write the header and frame data to a file. Then use the DBiBrowse API to display the contents of the file. See “Chapter 14. Detecting Video Scene Changes” on page 167 for detailed information about using the DBvGetNextFrame, DBvSeekNextFrame, and DBvFrameDatato24BitRGB APIs, and for further information about displaying a video frame.

Playing an audio or video

Use the DBaPlay API to play an audio stored in a database table. Use the DBvPlay API to play a video stored in a database table. See “Using the display or play APIs” on page 119 for detailed information on using these APIs.

Playing audio/video

Chapter 13. Querying Images by Content

Figure 25 shows an application program that allows users to search for images in a database using a visual example as search criteria, that is, an image that demonstrates a predominant color or texture pattern. With such an application, users can supply an image as input to the search. The application then matches the color or texture of the source image against those of the stored images, and returns the images whose color or texture most closely match the input.

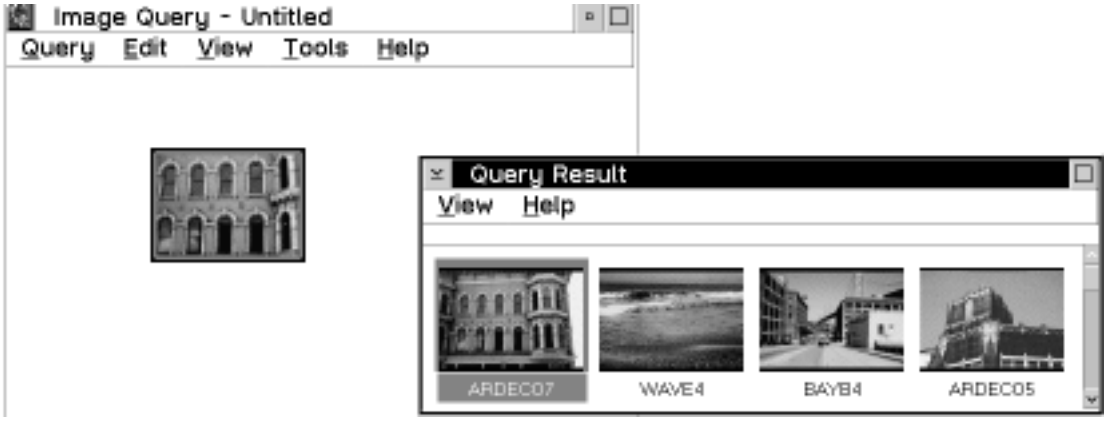


Figure 25. Query by image content. The color or texture of a visual example is used to search for images stored in a database table.

This capability to query images by their visual features is called **Query by Image Content (QBIC)** [5]. This chapter describes how to use APIs and UDFs provided with the Image Extender to build applications like the one just described. It also describes how to use commands and APIs provided with the Image Extender to perform QBIC administrative tasks.

How to query by image content

To query by image content:

1. Create a QBIC catalog for the images.
2. Catalog the images. This means adding entries for the images to the catalog and storing values for image features.

5. The Image Extender includes software developed by the University of California, Berkeley and its contributors.

See “QBIC catalogs” on page 20 for a description of QBIC catalogs and image features.

3. Build a query. The query identifies the features to be used as search criteria, their values, and their weights (that is, emphasis to be placed on each feature). You can specify these query attributes in a character string called a query string. Alternatively, you can create a query object and associate these attributes with the query object. You can then give the query object a name and save it.
4. Run the query. When you run the query, you specify a query string as input or you identify a query object for the query. In either case, you also identify the images to be searched. In either case, you can submit the query from the DB2 command line or from within a program.

In response, the Image Extender computes the feature values for the query. It compares the value to the feature values stored in the QBIC catalog for the target images. The Image Extender then computes a score that indicates how similar the feature values of each target image is to the source.

You can tell the Image Extender to return the images whose feature values are most similar to the source. You can also tell the Image Extender to return the scores of one or more images.

Managing QBIC catalogs

Before images can be queried by content, they must be cataloged in a QBIC catalog. A QBIC catalog holds data about the visual features of images.

You create a QBIC catalog for each column of images in a user table that you want to make available for querying by content. There can be no more than one QBIC catalog for each column of images in a user table, and multiple columns cannot share the same QBIC catalog.

When you create a QBIC catalog, you identify the features for which you want the Image Extender to store data. You also indicate whether you want the Image Extender to automatically catalog an image. Automatic cataloging means the Image Extender will automatically create entries for an image in the catalog when the image is stored in a user table. If the image is not automatically cataloged, you must manually catalog it. This means that you explicitly tell the Image Extender to create entries in the catalog for the image.

After you create a QBIC catalog, you can:

- Open the catalog for subsequent actions on it
- Change the setting for automatic cataloging to manual cataloging, or from manual to automatic

- Add a feature to the catalog
- Remove a feature from the catalog
- Retrieve information about the catalog, such as the name of the user table and column associated with the catalog, or the features for which data is stored in the catalog
- Manually catalog images in the catalog
- Uncatalog an image (that is, remove the entries for the image from the catalog)
- Recatalog images
- Redistribute the catalog (that is, when adding or dropping nodes in a partitioned database system)
- Close the catalog
- Delete the catalog

You can perform these tasks, including creating a QBIC catalog, in an application program using APIs provided by the Image Extender. You can also perform many of the tasks using the db2ext command-line processor.

Creating a QBIC catalog

Use the QbCreateCatalog API or the CREATE QBIC CATALOG command to create a QBIC catalog. To create the catalog, you must be the owner of the user table whose images will be cataloged. In addition, you must have CREATE TABLE authority for the database that will contain the catalog. The user table and image column must be enabled for the Image Extender before you create a QBIC catalog for the images in that column.

When you create a QBIC catalog, you:

- Name the user table and column that contain the images to be cataloged.
- Indicate whether images will be automatically cataloged when stored in a user table. Automatic cataloging means the Image Extender will catalog an image when the image stored in a user table. Manual cataloging means you explicitly request the Image Extender to catalog an image. (See “Manually cataloging an image” on page 134 for information on how to manually catalog an image.)

The user table and column must be enabled: The user table and the column must be enabled for the Image Extender before you create a QBIC catalog for the images in that column. (See “Chapter 6. Preparing Data Objects for Extender Data” on page 51 for information on enabling user tables and columns for the Image Extender.)

Using the API: When you use the QbCreateCatalog API, you indicate automatic or manual cataloging by specifying an auto-catalog value. A value of 1 indicates automatic cataloging; a value of 0 indicates manual cataloging.

For example, the following statements create a QBIC catalog for the images in the picture column of the employee table. The images will be automatically cataloged when they are stored in the employee table:

```
SQLINTEGER autoCatalog=1;                       /* automatic cataloging */

rc=QbCreateCatalog(
                  "employee",                   /* user table */
                  "picture",                    /* image column */
                  autoCatalog);                 /* auto catalog setting */
```

Using the command line: When you issue the CREATE QBIC CATALOG command, you indicate automatic cataloging by specifying ON, manual cataloging by specifying OFF. OFF is the default.

For example, the following command creates the same QBIC catalog as in the API example:

```
CREATE QBIC CATALOG employee picture on
```

Back up the QBIC catalog: The Image Extender stores a QBIC catalog in files. You should periodically back up these files in case you need to recover the catalog. In an AIX, HP-UX, or Sun Solaris server, the files are located in the /home/*instance_owner*/dmb/qbic directory, where, *instance_owner* is the user ID of the instance owner. In an OS/2 or Windows server, the files are located in the *destination*\instance*instance_name*\qbic directory, where *destination* is the directory where the Image Extender is installed, and *instance_name* is the name of the extender instance.

Opening a QBIC catalog

You need to open a QBIC catalog to perform subsequent actions that change the catalog or use information in the catalog. For example, you need to open a QBIC catalog before you add a feature to the catalog, or before you search for images by content (this accesses feature information in the catalog).

To open a QBIC catalog, use the QbOpenCatalog API call or OPEN QBIC CATALOG command. When you open a QBIC catalog, you:

- Name the user table and image column for the catalog.
- Specify the mode in which you want the catalog opened (this is implicit when you use the OPEN QBIC CATALOG command). You can open a catalog for operations that read from it, such as searching for images by content. Or you can open a catalog for operations that update it, such as adding a feature. You must have SELECT authority for the user table to

open the catalog for read operations. You must have UPDATE authority for the user table to open the catalog for update operations.

What if a catalog is already open? You cannot open a catalog for update operations if the catalog is open for update in another session. When you open a QBIC catalog, the Image Extender closes any QBIC catalog that you already opened in the current session.

Using the API: When you use the QbOpenCatalog API, you explicitly specify the mode in which you want the catalog opened. Specify:

- The API parameter qbiRead to open the catalog for operations that read from it.
- The API parameter qbiUpdate to open the catalog for operations that update it.

QbiRead and QbiUpdate are constants defined in the include (header) file for QBIC, dmbqbapi.h.

You also need to point to the catalog handle. The catalog handle has a QBIC-specific data type of QbCatalogHandle. This data type is also defined in dmbqbapi.h. The Image Extender returns the catalog handle value as output from the API.

For example, the following API call opens a QBIC catalog for operations that read from the catalog:

```
SQLINTEGER mode;
QbCatalogHandle  *CatHdl;

mode=qbiRead;                                   /* open catalog for */
                                                /* read operations */
rc=QbOpenCatalog(
                 "employee",                    /* user table */
                 "picture",                     /* image column */
                 mode,                          /* open catalog mode */
                 &CatHdl);                      /* catalog handle */
```

Using the command line: When you issue the OPEN QBIC CATALOG command, the Image Extender attempts to open the catalog for update operations. If the catalog is currently open for update in another session, the Image Extender opens the catalog for read operations.

For example, the following command opens a QBIC catalog; the Image Extender attempts to open it for update operations:

```
OPEN QBIC CATALOG employee picture
```

Close the catalog when you finish QBIC-related activities: When you open a QBIC catalog, the Image Extender allocates resources to it such as memory. Close the catalog when you finish QBIC-related activities. This frees up the allocated resources.

Changing the auto catalog setting

Use the QbSetAutoCatalog API or the SET QBIC AUTOCATALOG command to change from automatic cataloging to manual cataloging or from manual to automatic. The QBIC catalog must be open for update before you change the catalog setting.

The change is not retroactive: When you change the autocatalog setting, it applies only to images added to the user table column after the change. Images already stored in the user table column are not affected. For example, if you change the setting from manual cataloging to automatic cataloging, only images added to the user table column after the change will be automatically cataloged. If you want to catalog images already in the table column, you need to manually catalog them. (See "Manually cataloging an image" on page 134 for information on how to manually catalog an image.)

Using the API: When you use the QbSetAutoCatalog API, specify the handle of the QBIC catalog (this is returned when you open the catalog with the QbOpenCatalog API). Also specify an auto catalog value of 1 for automatic cataloging, or a value of 0 for manual cataloging.

In the following example, manual cataloging is specified for a QBIC catalog associated with the images in the picture column of the employee table. Notice that the QBIC catalog is first opened for operations that update it.

```
SQLINTEGER mode;
SQLINTEGER autoCatalog=0;                          /* manual cataloging */

QbCatalogHandle  *CatHdl;

mode=qbiUpdate;                                    /* open catalog for */
                                                   /* update */
/* Open a QBIC catalog */
rc=QbOpenCatalog(
                  "employee",                      /* user table */
                  "picture",                       /* image column */
                  mode,                            /* open catalog mode */
                  &CatHdl);                        /* catalog handle */

/* Change the auto catalog setting */
rc=QbSetAutoCatalog(
                   CatHdl,                         /* catalog handle */
                   autoCatalog);                   /* auto catalog flag */
```

Using the command line: When you issue the SET QBIC AUTOCATALOG command, you indicate automatic cataloging by specifying ON, manual cataloging by specifying OFF. The command acts on the currently open catalog.

For example, the following command sets automatic cataloging off for the currently open QBIC catalog:

```
SET QBIC AUTOCATALOG off
```

Adding a feature to a QBIC catalog

Use the QbAddFeature API or the ADD QBIC FEATURE command to add a feature to a QBIC catalog. You must add at least one feature to a QBIC catalog before you can catalog an image in it. The QBIC catalog must be open for update before you add a feature.

When you add a feature to a catalog, specify the name of the feature that you want to add (the feature names are listed in Table 8).

Table 8. QBIC Feature Names

Feature name	Feature
QbColorFeatureClass	Average color
QbColorHistogramFeatureClass	Histogram color
QbDrawFeatureClass	Postional color
QbTextureFeatureClass	Texture

You might have to recatalog images: If you add a feature to a QBIC catalog, the Image Extender will not automatically store data about the new feature for already cataloged images, even if automatic cataloging is set on. To include data about a new feature for already cataloged images, you need to recatalog the images (see “Recataloging images” on page 136).

Using the API: When you use the QbAddFeature API, you need to specify the handle of the QBIC catalog in addition to the feature name. Notice the use of the constant qbiMaxFeatureName for the length of the feature name. The constant is defined in the include (header) file for QBIC, dmbqbapi.h, as the value 50.

In the following example, the QbAddFeature API is used to add the histogram color feature to a QBIC catalog:

```
char  featureName[qbiMaxFeatureName];

QbCatalogHandle   CatHdl;

strcpy(featureName,"QbColorHistogramFeatureClass");
```

```
rc=QbAddFeature(
                CatHdl,                          /* catalog handle */
                featureName);                    /* feature name */
```

Using the command line: The ADD QBIC FEATURE command acts on the currently open catalog. In the following example, the command is used to add the positional color feature to the currently open catalog:

```
ADD QBIC FEATURE QbDrawFeatureClass
```

Removing a feature from a QBIC catalog

Use the QbRemoveFeature API or the REMOVE QBIC FEATURE command to remove a feature from a QBIC catalog. The Image Extender deletes the catalog table for the feature. As a result, data for that feature is not stored when you catalog an image. The QBIC catalog must be open for update before you remove a feature.

When you remove a feature from a catalog, specify the name of the feature that you want to remove.

Using the API: When you use the QbRemoveFeature API, you need to specify the handle of the QBIC catalog in addition to the feature name.

In the following example, the QbRemoveFeature API is used to remove the histogram color feature from a QBIC catalog:

```
char  featureName[qbiMaxFeatureName];

QbCatalogHandle   CatHdl;

strcpy(featureName,"QbColorHistogramFeatureClass");

rc=QbRemoveFeature(
                CatHdl,                          /* catalog handle */
                featureName);                    /* feature name */
```

Using the command line: The REMOVE QBIC FEATURE command acts on the currently open catalog. In the following example, the command is used to remove the positional color feature from the currently open QBIC catalog:

```
REMOVE QBIC FEATURE QbDrawFeatureClass
```

Retrieving information about a QBIC catalog

You can retrieve the following information about a QBIC catalog:

- The name of the user table and image column associated with the catalog.
- The number of features for which data is stored in the catalog, and their feature names.

- Whether the Image Extender automatically catalogs images when they are stored in the user table.

Use the QbGetCatalogInfo API to retrieve the user table and column names, the number of features, and auto catalog setting. Use the QbListFeatures API to retrieve the feature names. Or use the GET QBIC CATALOG INFO command to retrieve all the information.

The QBIC catalog must be open before you can retrieve information.

Using the API: When you use the QbGetCatalogInfo API, you need to specify the handle of the QBIC catalog. You also need to point to a structure in which the Image Extender returns the catalog information. The catalog information structure is defined in the include (header) file for QBIC, dmbqapi.h, as follows:

```
typedef struct{
          char       tableName[qbiMaxTableName+1]    /* user table */
          char       columnName[qbiMaxColumnName+1]  /* image column */
          SQLINTEGER featureCount;                   /* number of features */
          SQLINTEGER autoCatalog;                    /* auto catalog flag */
} QbCatalogInfo;
```

When you issue the QbListFeatures API call, you need to allocate a buffer to hold the returned feature names. Feature names stored in the buffer are separated by a blank character. You also need to specify the catalog handle, and the size of the buffer for the returned feature names. To estimate the needed buffer size, you can use the feature count returned by the QbGetCatalogInfo API, and multiply the count by the longest feature name. You can use the constant qbiMaxFeatureName as the size of the longest feature name.

The API calls in the following example retrieve information about a QBIC catalog. Notice how the feature count returned by the QbGetCatalogInfo API and the qbiMaxFeature name constant are used to calculate the buffer size for the QbListFeatures API:

```
long  bufSize;
long  count;
char  *featureNames;

QbCatalogHandle  CatHdl;
QbCatalogInfo    catInfo;

/* Get user table name, image column name, feature count, */
/* and auto catalog setting */

rc=QbGetCatalogInfo(
               CatHdl,                          /* catalog handle */
               &catInfo);                       /* catalog info. structure */
```

```
/* List feature names */

bufSize=catInfo.featureCount*qbiMaxFeatureName;
featureNames=malloc(bufSize);

rc=QbListFeatures(
                CatHdl,                          /* catalog handle */
                bufSize                          /* size of buffer */
                count,                           /* feature count */
                featureNames);                   /* buffer for feature names */
```

Using the command line: The GET QBIC CATALOG INFO command acts on the currently open catalog. In the following example, the command is used to retrieve information about the currently open QBIC catalog:

```
GET QBIC CATALOG INFO
```

Manually cataloging an image

When you create a catalog, you indicate whether you want the Image Extender to automatically catalog an image when the image is stored in a user table. If an image is not automatically cataloged, you must manually catalog it after it is stored in the user table. You can manually catalog a single image or an entire column of images.

Manually cataloging a single image

Use the QbCatalogImage API to manually catalog a single image. You cannot catalog an image by command, because there is no way to identify the individual image on the command line. When you use the API, specify the catalog handle and the image handle (you can retrieve the image handle from the user table). The QBIC catalog must be open before you manually catalog an image.

For example, the following statements retrieve an image handle from a user table and then catalog the image:

```
/* Retrieve the image handle */

EXEC SQL BEGIN DECLARE SECTION;
char Img_hdl[251];
EXEC SQL END DECLARE SECTION;

QbCatalogHandle   CatHdl;

EXEC SQL SELECT PICTURE INTO :Img_hdl
  FROM EMPLOYEE
  WHERE NAME='Anita Jones';

/* Catalog the image*/
```

```
rc=QbCatalogImage(
                CatHdl,                          /* catalog handle */
                Img_hdl);                        /* image handle */
```

Manually cataloging a column of images

Use the QbCatalogColumn API or the CATALOG QBIC COLUMN command to manually catalog a column of images. The Image Extender catalogs only images in the column that are currently not cataloged, and it catalogs those images for all features in the catalog. The QBIC catalog must be open for update before you manually catalog a column of images.

Using the API: When you use the QbCatalogColumn API, specify the catalog handle. The Image Extender uses the images in the user table column associated with the specified catalog.

For example, the following API call catalogs the uncataloged images in a user table column associated with the specified catalog. The images are cataloged for all the features in the catalog:

```
QbCatalogHandle   CatHdl;

rc=QbCatalogColumn(
                CatHdl);                         /* catalog handle */
```

Using the command line: Use the CATALOG QBIC COLUMN command to manually catalog a column of images. You can also use the command to recatalog images (see “Recataloging images” on page 136). Specify the parameters FOR and NEW. (FOR and NEW are default parameters.)

In the following example, the command is used to catalog the uncataloged images in the table column associated with the currently-opened catalog. The images are cataloged for all the features in the catalog:

```
CATALOG QBIC COLUMN FOR NEW
```

Uncataloging an image

Uncataloging an image means removing entries for the image from a QBIC catalog. Use the QbUncatalogImage API to uncatalog an image. You cannot uncatalog an image by command, because there is no way to identify the individual image on the command line. When you use the API, specify the catalog handle and the image handle (you can retrieve the image handle from the user table). The QBIC catalog must be open for update before you uncatalog an image.

For example, the following statements retrieve an image handle from a user table and then uncatalog the image:

```
/* Retrieve the image handle */

EXEC SQL BEGIN DECLARE SECTION;
char Img_hdl[251];
EXEC SQL END DECLARE SECTION;

QbCatalogHandle   CatHdl;

EXEC SQL SELECT PICTURE INTO :Img_hdl
  FROM EMPLOYEE
  WHERE NAME='Anita Jones';

/* Uncatalog the image */

rc=QbUncatalogImage(
               CatHdl,                          /* catalog handle */
               Img_hdl);                        /* image handle */
```

Recataloging images

When you catalog an image, the Image Extender analyzes the features of the image that were identified to the QBIC catalog and stores values for those features in the catalog. When you add a feature to a QBIC catalog, the Image Extender does not automatically analyze the new feature for already cataloged images. To add values for the new feature to the catalog, you need to recatalog the images.

Use the QbReCatalogColumn API or the CATALOG QBIC COLUMN command to recatalog the images in a QBIC catalog. The Image Extender analyzes the new feature for the images and reanalyzes the existing features for the images. The QBIC catalog must be open before you recatalog images.

Using the API: When you use the QbReCatalogColumn API, specify the catalog handle.

In the following example, the images in a QBIC catalog are reanalyzed:

```
QbCatalogHandle   CatHdl;

rc=QbReCatalogColumn(
               CatHdl);                         /* catalog handle */
```

Using the command line: Use the CATALOG QBIC COLUMN command to recatalog images. The command acts on the currently open catalog. You can also use the command to manually catalog images (see “Manually cataloging an image” on page 134).

When you issue the command, specify the parameters FOR and ALL. This tells the Image Extender that you want to recatalog all the images.

In the following example, the cataloged images in the currently-opened QBIC catalog are recataloged:

```
CATALOG QBIC COLUMN FOR ALL
```

Redistributing a QBIC catalog (EEE Only)

Use the DMBRedistribute API or the REDISTRIBUTE NODEGROUP command to redistribute QBIC feature data when a node is added to or removed from a nodegroup. The command places the QBIC feature data on the same node as the corresponding user data.

If the redistribution process returns an error, you can re-run the command with or without the CONTINUE parameter according to the instructions provided by the command response. This option instructs the system to continue from where it stopped, rather than starting from the beginning. The CONTINUE parameter should not be used the first time you run the REDISTRIBUTE NODEGROUP command after running DB2's REDISTRIBUTE command.

To maintain data integrity, redistribute one nodegroup at a time. Wait until one nodegroup has finished redistribution before starting another.

Using the API: The following example shows how to redistribute QBIC feature data in the nodegroup called groupone:

```
#include <dmbrdst.h>

rc = DMBRedistribute(groupone,"continue");
```

Using the command line: The following example displays how the REDISTRIBUTE NODEGROUP command is used to redistribute data for the node called my_nodegroup using the CONTINUE parameter:

```
redistribute nodegroup my_nodegroup continue
```

Closing a QBIC catalog

Use the QbCloseCatalog API or the CLOSE QBIC CATALOG command to close a QBIC catalog. The catalog must be open before you close it.

Using the API: When you issue the QbCloseCatalog API call, specify the catalog handle. For example:

```
QbCatalogHandle   CatHdl;

rc=QbCloseCatalog(
                CatHdl);                          /* catalog handle */
```

Using the command line: The CLOSE QBIC CATALOG command acts on the currently open catalog. In the following example, the command is used to close the currently open QBIC catalog:

```
CLOSE QBIC CATALOG
```

Deleting a QBIC catalog

Deleting a QBIC catalog deletes all the feature data in the catalog tables. As a result, the associated images are no longer available for querying by content. To delete a QBIC catalog, you must have ALTER or CONTROL authority for the table associated with the catalog. The catalog must be open before you delete it.

Use the QbDeleteCatalog API or DELETE QBIC CATALOG command to delete a QBIC catalog. When you delete a QBIC catalog, name the user table and column associated with the catalog.

Using the API: In the following example, the QbDeleteCatalog API is used to delete a QBIC catalog:

```
rc=QbDeleteCatalog(
                  "employee",                    /* user table */
                  "picture");                    /* image column */
```

Using the command line: The DELETE QBIC CATALOG command acts on the currently open catalog. In the following example, the command is used to delete the currently open QBIC catalog:

```
DELETE QBIC CATALOG employee picture
```

QBIC catalog sample program

Figure 26 on page 140 shows part of a program coded in C that creates a QBIC catalog. The program also catalogs into the QBIC catalog a column of images. You can find the complete program in the QBCATDMO.C file in the SAMPLES subdirectory. Before executing the complete program, you must execute the ENABLE and POPULATE sample programs (also found in the SAMPLES subdirectory). For more information about the sample programs, see “Appendix B. Sample Programs and Media Files” on page 559.

Note the following points in the program:

1. Include the dmbqbapi header file.
2. Connect to the database.
3. Create the catalog. The catalog is created with automatic cataloging turned off.
4. Open the catalog for update.

5 Add the average color feature to the catalog.
6 Catalog a column of images.
7 Close the catalog.

```
#include <sql.h>
#include <sqlcli.h>
#include <sqlcli1.h>
#include <dmbqbqpi.h> 1
#include <stdio.h>

/*****************************************************************************/
/*  Define the function prototypes                                           */
/*****************************************************************************/

void printError(SQLHSTMT hstmt);
void createCatalog();
void openCatalog();
void closeCatalog();
void addFeature();
void catalogImageColumn();

QbCatalogHandle cHdl = 0;

static SQLHENV henv;
static SQLHDBC hdbc;
static SQLHSTMT hstmt;
static SQLRETURN rc;
char tableName[] = "sobay_catalog";
char columnName[] = "covers";

SQLCHAR uid[18+1];
SQLCHAR pwd[30+1];
SQLCHAR dbnName[SQL_MAX_DSN_LENGTH+1];

void main ()
{
/*---- prompt for database name, userid, and password ----*/
  printf("Enter database name:\n");
  gets((char *) dbName);
  printf("Enter userid:\n");
  gets((char *) pwd);
/* set up the SQL CLI environment */
 SQlAllocEnv(&henv);
 SQLAllocConnect(henv, &hdbc);
 rc = SQLConnect(hdbc,dbname,SQL_NTS,uid,SQL_NTS,pwd,SQL_NTS); 2
 if (rc != SQL_SUCCESS)
 {
  printError(SQL_NULL_HSTMT);
  exit(1);
 }
```

Figure 26. QBIC catalog sample program (Part 1 of 4)

```
 createCatalog();
   openCatalog();
   addFeature();
 getCatalogInfo();
 listFeatures();
 catalogImageColumn();
 closeCatalog();

 SQLDisconnect(hdbc);
 SQLFreeConnect(hdbc);
 SQLFreeEnv(henv);
}
/*********************************************************************/
void createCatalog()
{
 SQLINTEGER autoCatalog = 0;
 SQLINTEGER retLen;
   SQLINTEGER errCode = 0;
   char errMsg[500];

   QbCreateCatalog( 3
         (char *) tableName,
         (char *) columnName,
         autoCatalog,
         0
        );

   DBiGetError(&errCode, errMsg);
   if(errCode) printf("Error code is %d Error Message %s", errCode, errMsg);
}
/*********************************************************************/
void openCatalog()
{
   SQLINTEGER errCode = 0;
   char errMsg[500];
 SQLINTEGER mode = qbiUpdate;

   QbOpenCatalog( 4
         (char *) tableName,
         (char *) columnName,
         mode,
         &cHdl
        );

   DBiGetError(&errCode, errMsg);
   if(errCode) printf("Error code is %d Error Message %s", errCode, errMsg);
}
```

Figure 26. QBIC catalog sample program (Part 2 of 4)

```
/*********************************************************************/
void addFeature()
{
 SQLINTEGER errCode=0;
 char errMsg[5
 if(cHdl) /* if we have an open catalog, else do nothing */
 {
   char featureName*lbrk.] = "QbColorFeatureClass";  5
     QbaddFeature(
         cHdl,
         featureName
        );

     DBiGetError(&errCode, errMsg);
     if(errCode) printf("Error code is %d Error Message %s", errCode, errMsg);
 }
 else
 {
  exit(1);
 }
}
/*********************************************************************/
void catalogImageColumn()
{
   SQLINTEGER errCode = 0;
   char errMsg[500];

 if(cHdl) /* if we have an open catalog, else do nothing */
 {
  SQLRETURN rc;
    QbCatalogColumn(  6
         cHdl,
        );

    DBiGetError(&errCode, errMsg);
    if(errCode) printf("Error code is %d Error Message %s", errCode, errMsg);

 else
 {
  exit(1);
 }
}
```

Figure 26. QBIC catalog sample program (Part 3 of 4)

```
/*********************************************************************/
void closeCatalog()
{
 if(cHdl) /* if we have an open catalog, else do nothing */
 {
    QbCloseCatalog( 7
         cHdl,
        );
 }
}
/*********************************************************************/
```

Figure 26. QBIC catalog sample program (Part 4 of 4)

Building queries

When you query images by content, you identify input for the query and a target set of cataloged images. The input for the query specifies the name of the features to be used in the query, feature values, and feature weights (that is, emphasis to be placed on each feature).

You have two ways to provide this input:

- Specify a query string in your query. The query string is a character string that specifies the features, feature values, and feature weights for the query.
- Create a query object and reference it in your query. The query object specifies features and feature weights. It also identifies a data source for each feature. The data source provides the value for each feature.

Specifying a query string

You can use a query string to identify the features, feature values, and feature weights for your query. A **query string** is a character string that has the form *feature_name value*, where *feature_name* is a QBIC feature name, and *value* is a value associated with the feature.

You can specify multiple features in a query. You then specify a feature name-value pair for each feature, as described in “Feature value” on page 144. Each pair is separated by the clause AND. When you specify multiple features in a query, you can also assign a weight to one or more of the features, as described in “Feature weight” on page 145. The query string then has the form *feature_name value weight*, where *weight* is the weight assigned to the feature.

The Image Extender provides an API (QbQueryStringSearch) and two UDFs (QbScoreFromStr and QbScoreTBFromStr) that use a query string. When you issue a query, you use the appropriate API or UDF and specify the query string as an input parameter. (See “Issuing queries by image content” on page 154 for details.)

Feature value

Specify a feature value in the query string for each feature in the query.

When passing a query inside a DB2 command, certain file-naming conventions must be followed for the query to function properly. Filenames that do not contain underscores can optionally be enclosed in double quotation marks. Filenames that do contain underscores must be enclosed in double quotation marks. Regardless of whether the filename includes underscores or not, each quotation mark surrounding a filename must be preceded by an escape character (\). If the query is not passed within a DB2 command, then there is no need to include escape characters with the quotation marks. See the example below:

```
db2 "select image_id from table
(mmdbsys.QbScoreTBFromStr
('QbTextureFeatureClass file=<server,\"patterns/ptrn07.gif\">',
'fabric',
'swatch_img',
10))
as T1"
```

Table 9 lists the values that you can specify for each feature. Directly below each feature name is a short version that can be used instead.

Table 9. Feature values that can be specified in query string

Feature name	Value
QbColorFeatureClass or AverageColor	color=<*Rvalue, Gvalue, Bvalue*> Each color value is an integer from 0 to 255 that identifies the red value (*Rvalue*), green value (*Gvalue*), and blue value (*Bvalue*) of the image. file=<*file_location, filename*> The (*file_location*) is the server file location. The *filename* is the complete file path specified in the format appropriate for the system in which the file resides. handle=<*image_handle*> The handle is the column in which the image resides.

Table 9. Feature values that can be specified in query string (continued)

Feature name	Value
QbColorHistogramFeatureClass or Histogram	histogram=<(*hist_value*, *Rvalue*, *Gvalue*, *Bvalue*>), ... Each histogram color value is specified in a clause that identifies the percent (1 to 100) of that color in the histogram (*hist_value*), and the red value (*Rvalue*), green value (*Gvalue*), and blue value (*Bvalue*) of that color. file=<*file_location*, *filename*> The (*file_location*) is the server file location. The *filename* is the complete file path specified in the format appropriate for the system in which the file resides. handle=<*image_handle*> The handle is the column in which the image resides.
QbDrawFeatureClass or Draw	file=<*file_location*, *filename*> handle=<*image_handle*> The (*file_location*) is the server file location. The *filename* is the complete file path specified in the format appropriate for the system in which the file resides. handle=<*image_handle*> The handle is the column in which the image resides.
QbTextureFeatureClass or Texture	file=<*file_location*, *filename*> handle=<*image_handle*> The (*file_location*) is the server file location. The *filename* is the complete file path specified in the format appropriate for the system in which the file resides. handle=<*image_handle*> The handle is the column in which the image resides.

Feature weight

If you specify multiple features in a query string, you can also specify a weight for one or more of the features. The weight of a feature indicates the emphasis that the Image Extender places on the feature when it computes similarity scores and returns results for a query by image content. The higher the weight you specify for a feature, the greater the emphasis on that feature in the query. The weight is a real number greater than 0.0, for example, 2.5 or 10.0. If you don't assign a weight in a query string, the Image Extender will

use the default weight for the feature. Assigning a weight has no meaning if that feature is the only feature specified in a query string. (That feature will always have full weight in the query.)

The weight for a feature is relative to other features specified in the query. For example, suppose you specify the average color and texture features in a query string, and also specify a weight value of 2.0 for average color. This tells the Image Extender to give the average color value twice the emphasis as the texture value.

Examples

The following query string specifies an average color of red:

```
QbColorFeatureClass color=<255, 0, 0>
```

The following query string specifies a histogram comprised of 10% red, 50% green, and 40% blue:

```
QbcolorHistogramFeatureClass histogram=<(10, 255, 0, 0), (50, 0, 255, 0),
                                        (40, 0, 0, 255)>
```

The following query string specifies an average color value and a texture value. The texture value is provided by an image in a server file. The weight of the texture is twice that of the average color:

```
QbColorFeatureClass color=<30, 200, 25> and
 QbTextureFeatureClass file=<server, "\patterns\pattern7.gif"> weight=2.0
```

Using a query object

You can use a query object to identify the features, feature values, and feature weights for your query. You create the query object and add features to it. Then you specify a data source for each feature. The data source provides a value for the feature. For example, a data source might be an image in a file. If average color is the pertinent feature, the average color of the image is associated with the query object. If you add multiple features to a query object, you can assign a weight to one or more of the features.

The Image Extender provides two APIs (QbQuerySearch and QbQueryNameSearch) and two UDFs (QbScoreFromName and QbScoreTBFromName) that use a query object. When you issue a query, you use the appropriate API or UDF and specify the query object as an input parameter. (See “Issuing queries by image content” on page 154 for details.)

Creating a query object

Use the QbQueryCreate API to create a query object.. In response, the Image Extender returns a handle for the query object.. The handle has a QBIC-specific data type of QbQueryHandle that is defined in the include (header) file for QBIC, dmbqbapi.h.

When you use the API, you need to point to the query object handle. You also need to specify the handle in APIs that perform other operations on the query object, such as adding a feature.

For example, the following API call creates a query object:

```
QbQueryHandle  qHandle;

rc=QbQueryCreate(
             &qHandle);                          /* query object handle */
```

Adding a feature to a query object

You identify the image features that you want the Image Extender to query by adding the features to a query object.

Use the QbQueryAddFeature API to add a feature to a query object. When you use the API, specify the query object handle. You also name the feature. You can specify only one feature in the API. You must issue a separate API call for each feature that you want to add to a query object.

In the following example, the QbQueryAddFeature API is used to add the average color feature to a query object:

```
char  featureName[qbiMaxFeatureName];
QbQueryHandle  qHandle;

rc=QbQueryAddFeature(
               qHandle,                          /* query object handle */
               "QbColorFeatureClass");           /* feature name */
```

Specifying the data source for a feature in a query object

Use the QbQuerySetFeatureData API to specify the data source for a feature in a query object. The data source can be:

- A cataloged or uncataloged image in a column of a user table
- An image file on a client workstation
- An image in a buffer on a client workstation

In addition, you can explicitly specify data for the average color or histogram color feature. For example, you can specify the red, green, and blue values of an average color.

When you use the API:

- Specify the query objecthandle.
- Name the feature.
- Point to the QbImageSource structure (see page 148 for details).

Using data source structures: Various structures are used to provide data source information for a query object. The structures are:

- QbImageSource
- QbColor
- QbHistogramColor

QbImageSource: The QbImageSource structure identifies the type of source for a feature in a query object. The structure is defined in the include (header) file for QBIC, dmbqbapi.h, as follows:

```
typedef struct{
          SQLINTEGER    type;
          union {
                char              imageHandle[MMDB_BASE_HANDLE_LEN+1];
                QbImageFile       clientFile;
                QbImageBuffer     buffer;
                QbSampleSource    reserved;
                QbColor           averageColor;
                QbHistogramColor  histogramColor[qbiHistogramCount];
          };
} QbImageSource;
```

The type field in the QbImageSource structure indicates the type of source. You can set the value in the field as follows:

Value	Meaning
qbiSource_ImageHandle	Source is in a user table column
qbiSource_ClientFile	Source is in a client workstation file
qbiSource_Buffer	Source is in a client workstation buffer
qbiSource_AverageColor	Source is an average color specification
qbiSource_HistogramColor	Source is a histogram color specification

These settings are valid only for the appropriate feature. For example, qbiSource_AverageColor is valid only for the average color feature.

Depending on the type of source, the Image Extender also examines other information that you specify. This is shown in Table 10 on page 149.

Table 10. What the Image Extender examines in QbImageSource

Source	What the Image Extender examines	Where specified
user table	image handle	image handle field of QbImageSource
file	name of file format of file	clientFile field of QbImageSource
buffer	name of file	QbImageBuffer (see page149 for details about using this structure)
average color specification	red, green, and blue color values	QbColor structure (see page 149 for details about using this structure)
histogram color specification	color values and percentages	QbHistogramColor structure (see page 149 for details about using this structure)

QbImageBuffer: Use the QbImageBuffer structure to specify the format, length, and content of an image when the data source is in a buffer. The structure is defined in the include (header) file for QBIC, dmbqbapi.h, as follows:

```
typedef struct{
          char          format[qbiImageFormatLength+1];
          SQLINTEGER    length;
          char*         image;
} QbImageBuffer;
```

QbColor: Use the QbColor structure to specify the red, green, and blue values of an average color when the data source is an average color specification. The structure is defined in the include (header) file for QBIC, dmbqbapi.h, as follows:

```
typedef struct{
          SQLUSMALLINT    red;          /*0 off - 65535 (fully on) */
          SQLUSMALLINT    green;        /*0 off - 65535 (fully on) */
          SQLUSMALLINT    blue;         /*0 off - 65535 (fully on) */
} QbColor;
```

Set the values in QbColor to indicate the amount of red, green, and blue pixels to be factored in the average value calculation. The values can range from 0 to 65535. A value of 0 means ignore the entry.

QbHistogramColor: Use the QbHistogramColor structure to specify each color component of a histogram color specification. The full specification for a histogram color is contained in an array of QbHistogramColor structures. Each structure contains a color value and a percentage. The color value is comprised of red, green, and blue pixel values. The percentage specifies the percentage of that color required in the target image.

The structure is defined in the include (header) file for QBIC, dmbqbapi.h, as follows:

```
typedef struct{
          QbColor         color;
          SQLUSMALLINT    percentage;   /*0 - 100 */
} QbHistogramColor;
```

Set the values in QbColor to indicate the amount of red, green, and blue pixels for the color. The values can range from 0 to 65535. Set the percentage to indicate the percentage of the specified color required in the target image. The value can range from 1 to 100. The sum of the percentages for the color components in a histogram color must be 100 or less.

Examples: The API in the following example specifies the data source for the histogram color feature in a query object. The data source is a file in the client workstation.

```
char           featureName[qbiMaxFeatureName];
QbQueryHandle  qHandle;
QbImageSource  imgSource;

imgSource.type=qbSource_ClientFile
strcpy(imgSource.clientFile.fileName,"/tmp/image.gif");
strcpy(imgSource.clientFile.format,"GIF");

rc=QbQuerySetFeatureData(
                qHandle,                              /* query object handle */
                "QbColorHistogramFeatureClass",       /* feature name */
                &imgSource);                          /* feature data source */
```

In the following example, the data source is an average color specification of red:

```
char                featureName[qbiMaxFeatureName];
QbColor             avgColor;
QbImageSource       imgSource;

imgSource.type=qbSource_AverageColor;
avgColor.red=255;
avgColor.green=0;
avgColor.blue=0;
strcpy(featureName,"QbColorFeatureClass");

rc=QbQuerySetFeatureData(
                qHandle,                              /* query object handle */
                featureName,                          /* feature name */
                &imgSource);                          /* feature data source */
```

Setting the weight of a feature in a query object

If you have added more than one feature to a query object, you can specify the weight that one or more features are to be given in a query. Use the

QbQuerySetFeatureWeight API to specify the weight of a feature. The weight of a feature indicates the emphasis that the Image Extender places on the feature when it computes similarity scores and returns results for a query by image content. The higher the weight you specify for a feature, the greater the emphasis on that feature in the query object.

You can specify a weight for one or more features in a query object, although you can specify a weight for only one feature each time you issue the QbQuerySetFeatureWeight API. If you don't assign a weight to a feature in a query object, the Image Extender will use the default weight for the feature. Assigning a weight to a feature has no meaning if that feature is the only feature in a query object. (That feature will always have full weight in the query object.)

When you use the API:

- Specify the query object handle.
- Specify the feature name.
- Point to the feature weight. You can set the weight to a real number greater than 0, for example, 2.5 or 10.0. The higher the value you specify, the greater the emphasis on that feature. The setting changes any weight previously set for the feature in the query object.

In the following example, a query object contains the average color feature and at least one other feature. The QbQuerySetFeatureWeight API is used to specify a weight for the average color feature in the query object:

```
char                featureName[qbiMaxFeatureName];
double              weight;
QbQueryObjectHandle qoHandle;

strcpy(featureName,"QbColorFeatureClass");
weight=5.00;

rc=QbQuerySetFeatureWeight(
                qoHandle,                       /* query object handle */
                featureName,                    /* feature name */
                &weight);                       /* feature weight */
```

Naming and saving a query object

Query objects are transient unless you save them. They exist only during a single database connection. Saving a query object allows you to use it even after the current database connection is dropped, or across program invocations.

The Image Extender provides two APIs (QbQuerySearch and QbQueryNameSearch) and two UDFs (QbScoreFromName and QbScoreTBFromName) that use a query object as input to a query by image

content (see “Issuing queries by image content” on page 154). The QbQuerySearch API requires you to identify the query object by its handle. The QbQueryNameSearch API, the QbScoreFromName UDF, and QbScoreTBFromName UDF require you to identify the query object by its name.

The query object handle is returned when you create a query (as described in “Creating a query object” on page 147). Use the QbQueryNameCreate API to name a query object. In response to the QbQueryNameCreate API call, the Image Extender saves the query object, by name, in the currently connected database.

When you issue the QbQueryNameCreate API, specify the query object handle and a query object name. The name can be up to 18 characters. You can also specify a description of up to 250 characters.

In the following example, the QbQueryNameCreate API is used to name and save a query object:

```
QbQueryHandle  qHandle;
char           qName[18];
char           qDesc[250];

strcpy(qName,"average_red");
strcpy(qDesc,"average color query, created 10/15/96");

rc=QbQueryNameCreate(
                qHandle,                          /* query object handle */
                qName,                            /* query object name */
                qDesc);                           /* query object description */
```

Retrieving information about a query object

You can determine what features (if any) have been added to a query object. You can also determine the current weight of a feature.

Use this API	To retrieve
QbQueryGetFeatureCount	The number of features in a query object
QbQueryListFeatures	The names of the features in a query object
QbQueryGetFeatureWeight	The weight of a feature in a query object

When you issue the QbQueryGetFeatureCount API, specify the query object handle. You also need to point to a counter. The Image Extender returns the feature count in the counter.

In the following example, the QbQueryGetFeatureCount API is used to determine the number of features in a query object:

```
SQLINTEGER     count;
QbQueryHandle  qHandle;

rc=QbQueryGetFeatureCount(
                qHandle,                          /* query object handle */
                &count);                          /* feature count */
```

When you issue the QbQueryListFeatures API call, you need to allocate a buffer to hold the returned feature name. You also need to specify the catalog handle, and the size of the buffer for the returned feature name.

In the following example, the QbQueryListFeatures API is used to retrieve the name of each feature in a query object:

```
SQLINTEGER     retCount,bufSize;
char*          featureName;
QbQueryHandle  qHandle;

bufSize=qbiMaxFeatureName;
featureName=(char*)malloc(bufSize);

rc=QbQueryListFeatures(
                qHandle,                          /* query object handle */
                bufSize                           /* size of buffer */
                &retCount,                        /* feature count */
                featureName);                     /* buffer for feature names */
```

Use the QbQueryGetFeatureWeight API to retrieve the weight setting for a feature added to a query object. If no weight was set, the API returns the default weight for the feature.

When you issue the QbQueryGetFeatureWeight API call:

- Specify the query object handle.
- Identify the feature name.
- Point to the weight variable. The Image Extender sets the variable to the weight setting, or to the default if no weight was set for the feature.

In the following example, the QbQueryGetFeatureWeight API is used to get the weight for the texture feature in a query object.

```
char                 featureName[qbiMaxFeatureName];
double               weight;
QbQueryObjectHandle  qoHandle;

strcpy(featureName,"QbTextureFeatureClass");

rc=QbQueryGetFeatureWeight(
                qoHandle,                         /* query object handle */
                featureName,                      /* feature name */
                &weight);                         /* feature weight */
```

Removing a feature from a query object

Remove a feature from a query object with the QbQueryRemoveFeature API. When you use the API, specify the query object handle and name the feature.

In the following example, the QbQueryRemoveFeature API is used to remove the histogram color feature from a query object:

```
char           featureName[qbiMaxFeatureName];
QbQueryHandle  qHandle;

strcpy(featureName,"QbColorHistogramFeatureClass");

rc=QbQueryRemoveFeature(
                qHandle,                          /* query object handle */
                featureName);                     /* feature name */
```

Deleting a query object

Delete a named query object with the QbQueryNameDelete API. Delete an unnamed query object with the QbQueryDelete API. The Image Extender deletes the query from the currently connected database.

When you use the QbQueryNameDelete API, specify the query object name. When you use the QbQueryDelete API, specify the query object handle.

In the following example, the QbQueryNameDelete API is used to delete a named query object:

```
char qName[18];

strcpy(qname, "average_red");

rc=QbQueryNameDelete(
                qName);                           /* query object name */
```

In the following example, the QbQueryDelete API is used to delete an unnamed query object:

```
QbQueryHandle  qHandle;

rc=QbQueryDelete(
            qHandle);                          /* query object handle */
```

Issuing queries by image content

After you catalog images, you can query one or more of the images by content. When you query an image by content, you identify input for the query and a target set of cataloged images. You can specify the input in a query string (see “Specifying a query string” on page 143) or in a query object (see “Using a query object” on page 146).

If you use a query string, you can submit the query from the DB2 command line or from within a program. If you use a query object, how you reference the query object determines whether you can submit the query from a DB2 command line, from within a program, or both. If you name and save a query object, you can then reference the query object by name in a UDF that accepts a named query (QbScoreFromName UDF or QbScoreTBFromName UDF). You can issue the UDF from the DB2 command line or from within a program. If you reference a query object by its handle, you can submit the query from within a program only.

The Image Extender compares the feature values specified in the query to those of the target images, and computes a score for each image. The score indicates how similar the feature values of the target image are to feature value specified in the query.

You can retrieve images whose feature values are most similar to the query. You can also query a single cataloged image and get its score, or get scores for all the cataloged images in a table column.

Querying images

The Image Extender provides three APIs to query the cataloged images in a table column. The APIs differ only in whether they require a query string or query object as input:

API	Input
QbQueryStringSearch	Query string
QbQuerySearch	Query object handle
QbQueryNameSearch	Query object name

In all three APIs, you also:

- Name the table and column that contains the images to be searched. The images must be cataloged in a QBIC catalog.
- Specify the maximum number of results to be returned.
- Point to a structure that specifies the scope of the query. Set the pointer to 0, a NULL value, or an empty string. This specifies that all cataloged images in the table column are searched.
- Specify the constant qbiArray to indicate that the results are stored in an array. The qbiArray constant is defined in the include (header) file for QBIC, dmbqbapi.h.

You also point to an array of output structures to contain the results of the search. In response, the Image Extender returns in these structures the handles of the target images whose feature values are most similar to the feature value of the query. It also returns a score for each image that indicates how similar

the feature value of the image is to the query. The structure is defined in the include (header) file for QBIC, dmbqbapi.h, as follows:

```
typedef struct{
          char       imageHandle[MMDB_BASE_HANDLE_LEN+1];
          SQLDOUBLE  SCORE
} QbResult;
```

You must allocate an array large enough to hold the maximum number of results you specify, and point to the array in the API. You must also point to a counter; the Image Extender sets the value of the counter to the number of results it returns.

In the following example, the QbQueryStringSearch API is used to query by content the cataloged images in a table column. Notice that the pointer to the query scope is set to a zero value.

```
QbResult       returns[MaxQueryReturns];
SQLINTEGER     maxResults=qbiMaxQueryReturns;
SQLINTEGER     count;
QbQueryHandle  qHandle;
QbResult       results[qbiMaxQueryReturns];

rc=QbQueryStringSearch(
          "QbColorFeatureClass color=<255, 0, 0>" /*query string */
          "employee",                        /* user table */
          "picture",                         /* image column */
          maxResults,                        /* maximum number of results */
          0,                                 /* query scope pointer * /
          qbiArray,                          /* store results in an array */
          &count,                            /* count of returned images */
          results);                          /* array of returned results */
```

Here is a request using the QbQuerySearch API. Notice that the query object handle is specified as input.

```
QbResult       returns[MaxQueryReturns];
SQLINTEGER     maxResults=qbiMaxQueryReturns;
SQLINTEGER     count;
QbQueryHandle  qHandle;
QbResult       results[qbiMaxQueryReturns];

rc=QbQuerySearch(
                qHandle,                      / query object handle */
                "employee",                   /* user table */
                "picture",                    /* image column */
                maxResults,                   /* maximum number of results */
                0,                            /* query scope pointer * /
                qbiArray,                     /* store results in an array */
                &count,                       /* count of returned images */
                results);                     /* array of returned results */
```

Here is a request using the QbQueryNameSearch API. Notice that the query object name (rather than the query object handle) is specified as input.

```
QbQueryHandle  qHandle;
char           qName[18];
char           qDesc[250];
char           results[qbiMaxQueryReturns];
SQLINTEGER     maxResults=qbiMaxQueryReturns;

/* Name the query object*/

strcpy(qName,"fshavgcol");
strcpy(qDesc,"average color query, created 10/15/96");

rc=QbQueryNameCreate(
                qHandle,                        /* query object handle */
                qName,                          /* query object name */
                qDesc);                         /* query object description */

/* Issue the query */

rc=QbQueryNameSearch(
                qName,                          /* query object name */
                "employee",                     /* user table */
                "picture",                      /* image column */
                maxResults,                     /* maximum number of results */
                0,                              /* query scope pointer * /
                qbiArray,                       /* store results in an array */
                &count,                         /* count of returned images */
                results);                       /* array of returned results */
```

Retrieving an image score

The Image Extender provides four UDFs that you can use in an SQL statement to retrieve the score of a cataloged image in a table column. The score is a double-precision, floating point value from 0.0 to a very large number approaching infinity. The lower the score, the closer the feature values of the image matches the feature values specified in the query. A score of 0.0 means that the image is an exact match.

The UDFs are:

- QbScorefromStr
- QbScoreTBfromStr
- QbScoreFromName
- QbScoreTBFromName

Use the QbScoreFromStr UDF or QbScoreFromName UDF to get the score of a single cataloged image. Use the QbScoreTBFromStrUDF or the QbScoreTBFromName UDF to get the score of multiple cataloged images in a table column.

Retrieving the score of a single image

Use the QbScoreFromStr UDF or QbScoreFromName UDF to get the score of a single cataloged image in a table column. Specify a query string as input to the QbScoreFromStr UDF. Specify the name of a query object as input to the QbScoreFromName UDF. With either UDF, you also specify the name of the table column that contains the target image.

In the following query, the QbScoreFromStr UDF is used to find the cataloged images in a table column whose average color score is very close to red.

```
SELECT name, description
FROM fabric
WHERE (QbScoreFromStr(
            'QbColorFeatureClass color=<255, 0, 0>',    /* query string *
                 swatch_img))>0.9                     /* table column */
```

In the following C programming example, the QbScoreFromName UDF is used to make a similar request. Notice that a query object name is specified as an input parameter (rather than a query string):

```
EXEC SQL SELECT NAME, DESCRIPTION
 INTO :hvName, :hvDesc
 FROM FABRIC
 WHERE (QBSCOREFROMNAME(
            'fshavgcol',                                 /* query object name *
             SWATCH_IMG))<0.1;                          /* table column */
```

You can also use QbScore, but it's deprecated: DB2 extenders Version 1 provided a UDF named QbScore to get the score of an image. The input parameters for the QbScore UDF and its results are the same as for the QbScoreFromName UDF. The QbScore UDF is accepted in DB2 extenders Version 5, however the preferred form is QbScoreFromName.

Retrieving the score of multiple images

Use the QbScoreTBFromStr UDF or QbScoreTBFromName UDF to get the score of multiple cataloged images in a table column. Both UDFs return a two-column table of image handles and scores; the rows in the table are in descending order of score. The name of the handle column in the result table is IMAGE_ID, the name of the score column is SCORE.

Specify a query string as input to the UDF. Specify the name of a query object as input to the QbScoreTBFromName UDF. With either UDF, you also specify the name of the table and column that contains the target images. You can also specify the maximum number of rows to return in the result table. If you do not specify a maximum number of results, the UDF will return a row for each cataloged image in the target table column.

In the following query, the QbScoreTBFromStr UDF is used to find the ten cataloged images in a table column whose texture is closest to that of an image in a server file.

```
SELECT name, description
FROM fabric
WHERE CAST (swatch_img as varchar(250))IN
 SELECT CAST (image_id as varchar(25)) FROM TABLE
 (QbScoreTBFromStr
 (QbTextureFeatureClass file=<server,"patterns/ptrn07.gif">'   /*query string */

      'fabric',                                                /* table */
      'swatch_img',                                            /* table column */
      10))                                        /* maximum number of results */
AS T1));
```

In the following C programming example, the QbScoreTBFromName UDF is used. Notice that a query object name is specified as an input parameter (rather than a query string). Also notice that no input is specified for the maximum number of results; the UDF will return a result for each cataloged image in the target table column:

```
EXEC SELECT NAME, DESCRIPTION
 INTO :hvName, :hvDesc
 FROM FABRIC
 WHERE CAST (swatch_img as varchar(250)) IN
(SELECT CAST (image_id as varchar(250)) FROM TABLE
(QBSCORETBFROMNAME
            'fstxtr',                                  /* query object name */
            'FABRIC',                                  /* table */
            'SWATCH_IMG'))                             /* table column */
AS T1));
```

QBIC query sample program

Figure 27 on page 161 shows part of a program coded in C that builds and runs a QBIC query. The code in the figure queries images by average color. It prompts the user to enter the name of a color or image file. The user can also use an image returned by a query as an example image for a subsequent query. The program then uses the named color or the color of the image as the average color to query a column of images.

You can find the complete program in the QBICDEMO.C file in the SAMPLES subdirectory. The complete program can be used to query images by histogram color or positional color as well as by average color. To run the complete program, you must execute the ENABLE, POPULATE, and QBCATDMO sample programs (also in the SAMPLES subdirectory). For more information about the sample programs, see “Appendix B. Sample Programs and Media Files” on page 559.

Note the following points in Figure 27 on page 161:

1 Include the dmbqbapi header file.

2 Prompt the user for database information.

3 Connect to the database.

4 Create a queryobject.

5 Add a feature to the query object.

6 Prompt the user for the type of input (color name, image file, or previously retrieved image).

7 Specify the data source for the feature. The data source is an explicit specification for average color.

8 Issue the query. The Image Extender searches the entire column of images. It also specifies 10 as the maximum number of images to be returned.

9 Display the next image in the set of returned images. For further information on displaying images, see “Displaying a full-size image or video frame” on page 123.

10 Delete the query object.

The SAMPLES subdirectory includes another program that demonstrates how to build and use a QBIC query. The program, in file QBICPICK.C, gives the user a way to graphically specify the search criteria for a QBIC query. For example, the program presents a color selector to choose average color. The program converts the selection to a query string.

```
#include <sql.h>
#include <sqlcli.h>
#include <sqlcli1.h>
#include <dmbqbqpi.h>  1
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <color.h>
#include <ctype.h>

#define   MaxQueryReturns        10

#define   MaxDatabaseNameLength  SQL_SH_IDENT
#define   MaxUserIdLength        SQL_SH_IDENT
#define   MaxPasswordLength      SQL_SH_IDENT
#define   MaxTableNameLength     SQL_LG_IDENT
#define   MaxColumnNameLength    SQL_LG_IDENT

static  char       databaseName[MaxDatabaseNameLength+1];
static  char       userid[MaxUserIdLength+1];
static  char       password[MaxPasswordLength+1];

static  char       tableName[MaxTableNameLength+1];
static  char       columnName[MaxColumnNameLength+1];

static  char       line[4000];

static  QbResult   results[MaxQueryReturns];
static  long       currentImage   = -1;
static  long       imageCount  = 0;

static  char*      tableAndColumn;

static  QbQueryHandle averageHandle = 0;
static  QbQueryHandle histogramHandle = 0;
static  QbQueryHandle drawHandle = 0;
static  QbQueryHandle lastHandle = 0;

static  SQLHENV    henv;
static  SQLHDBC    hdbc;
static  SQLHSTMT   hstmt;
static  SQLRETURN  rc;

static  char*    listQueries =
"SELECT NAME,DESCRIPTION FROM MMDBSYS.QBICQUERIES ORDER BY NAME";

static  char*    menu[] = {
```

Figure 27. QBIC query sample program (Part 1 of 6)

```
/*
 12345678901234567890123456789012345678901234567890123456789012345678901234567890 */
"",
"+-------------------------------------------------------------------------------+",
"| AVERAGE COLOR colorname                                                       |",
"| AVERAGE FILE filename format                                                  |",
"| AVERAGE LAST                                                                  |",
"| Press Enter to display the next image in the series                           |",
"+-------------------------------------------------------------------------------+",
"",
0
};

static  char*       help[] = {
"",
"AVERAGE     Execute an average color query",
"  COLOR       Specifies the color to query for",
"  FILE        Specifies the file to compute the average color from",
"  LAST        Specifies the last displayed image be used to compute the color",
"  NEXT        Displays the next image from the current query or nothing if",
"                all of the image have been displayed."
"",
">>pause<<",
0
};
/*****************************************************************************/
/* doNext()                                                                  */
/*****************************************************************************/
static void doNext(void)
{
int ret;
   if (currentImage < imageCount)
      currentImage++;
   if (currentImage < imageCount)
      ret = DBiBrowse("/usr/local/bin/xv %s", MMDB_PLAY_HANDLE,
            results[currentImage].imageHandle, MMDB_PLAY_NO_WAIT);  9
}
```

Figure 27. QBIC query sample program (Part 2 of 6)

```
/*****************************************************************************/
/* doAverage()                                                               */
/*****************************************************************************/
static void doAverage(void)
{
   QbQueryHandle qohandle = 0; QbImageSource is; char* type;
   char* arg1; char* arg2;

   type = nextWord(0);
   if (abbrev(type, "color", 1)) {
      is.type = qbiSource_AverageColor;
      arg1 = nextWord(0);
      if (arg1 == 0) {
         printf("AVERAGE COLOR command requires a colorname argument.\n");
         return;
      }
      if (getColor(arg1, &is.averageColor) == 0) {
         printf("The colorname entered was not recognized.\n");
         return;
      }
   }
   else if (abbrev(type, "file", 1)) {
      is.type = qbiSource_ClientFile;
      arg1 = nextWord(0);
      if (arg1 == 0) {
         printf("AVERAGE FILE command requires a filename argument.\n");
         return;
      }
      arg2 = nextWord(0);
      if (arg2 == 0) {
         printf("AVERAGE FILE command requires a file format argument.\n");
         return;
      }
      strcpy(is.clientFile.fileName, arg1);
      strcpy(is.clientFile.format, arg2);
   }
   else if (abbrev(type, "last", 1)) {
      is.type = qbiSource_ImageHandle;
      if (0 <= currentImage &&; currentImage < imageCount)
         strcpy(is.imageHandle, results[currentImage]imageHandle);
      else {
         printf("No last image for AVERAGE LAST command\n");
         return;
      }
   }
```

Figure 27. QBIC query sample program (Part 3 of 6)

```
   else {
      printf("AVERAGE command only supports COLOR, FILE, and LAST types.\n");
      return;
   }

   _QbQuerySetFeatureData(averageHandle, "QbColorFeatureClass", &is);  7
   _QbQuerySearch(averageHandle, tableAndColumn, "IMAGE",
         MaxQueryReturns, 0, 0, &imageCount, results);  8
   lastHandle = averageHandle;

   currentImage = -1;
}
/*****************************************************************************/
/* commandLoop()                                                             */
/*****************************************************************************/
void commandLoop(void)
{
   int done = 0;

   while (!done) {  6
      displayText(menu);
      printf("%d", currentImage + 1);
      if (0 <= currentImage &&; currentImage < imageCount)
            printf("  %8.6f", results[currentImage].score);
      printf("> ");
      gets(line);
      done = processCommand(line);
   }
}
```

Figure 27. QBIC query sample program (Part 4 of 6)

```
/*****************************************************************************/
/* main()                                                                    */
/*****************************************************************************/
void main(void)
{
   char* inst;
   int i;

   printf("\n\n");
   printf("Please enter: database_name [user_id] [password] "\n"); 2
   gets(line);
   if (copyWord(line, databaseName, sizeof(databaseName)) == 0)
      exit(0);
   copyWord(0, userid, sizeof(userid));
   copyWord(0, password, sizeof(password));
   printf("\n");

   if (SQLAllocEnv(&henv) != SQL_SUCCESS)
      sqlError(SQL_NULL_HSTMT);
   if (SQLAllocConnect(henv, &hdbc) != SQL_SUCCESS)
      sqlError(SQL_NULL_HSTMT);
   if (SQLConnect(hdbc, 3
                  (SQLCHAR*)databaseName,
                  SQL_NTS,
                  (SQLCHAR*)userid,
                  SQL_NTS,
                  (SQLCHAR*)password,
                  SQL_NTS) != SQL_SUCCESS)
      sqlError(SQL_NULL_HSTMT);

   printf("Initializing . . .\n");
```

Figure 27. QBIC query sample program (Part 5 of 6)

```
   inst = getenv("DB2INSTANCE");
   if (inst != 0 &&; strcmp(inst, "keeseyt") == 0)
      tableAndColumn = "KEESEY.TEST";
   else
      tableAndColumn = "QBICDEMO.TEST";

   _QbQueryCreate(&averageHandle); 4
   _QbQueryAddFeature(averageHandle, "QbColorFeatureClass");
   _QbQueryCreate(&histogramHandle);
   _QbQueryAddFeature(histogramHandle, "QbColorHistogramFeatureClass");
   _QbQueryCreate(&drawHandle);
   _QbQueryAddFeature(drawHandle, "QbDrawFeatureClass"); 5

   commandLoop();

   _QbQueryDelete(drawHandle);
   _QbQueryDelete(histogramHandle); 10
   _QbQueryDelete(averageHandle);

   if (SQLDisconnect(hdbc) != SQL_SUCCESS)
      sqlError(SQL_NULL_HSTMT);
   if (SQLFreeConnect(hdbc) != SQL_SUCCESS)
      sqlError(SQL_NULL_HSTMT);
   if (SQLFreeEnv(henv) != SQL_SUCCESS)
      sqlError(SQL_NULL_HSTMT);
}
```

Figure 27. QBIC query sample program (Part 6 of 6)

Chapter 14. Detecting Video Scene Changes

This chapter describes how to detect scene changes in a video clip with the DB2 Video Extender APIs. These APIs are available in all DB2 Video Extender platforms except Windows 3.1. Video scene change detection is supported for video clips in MPEG-1 format only.

What is a video scene change?

Imagine a television studio that records programs on video tape for subsequent telecast. Recently the studio has started to store clips of its video tapes in a DB2 database using the Video Extender. This gives studio personnel the opportunity to query traditional types of information about their programs as well as view clips of the programs.

The studio would like the option of previewing a video clip. They want the ability to view a visual summary, called a storyboard. An example of a storyboard is shown in Figure 28 on page 168. Viewing storyboards can help studio employees get the gist of a video without having to view the entire video. It can also help employees decide if a video is the right video for their needs (for example, whether it merits downloading and viewing). This requirement is very important to the studio. Viewing a storyboard instead of an entire video can reduce download and viewing time dramatically. For more information on using video scene change detection capabilities in this way, see “Storing information about all the shots in a video” on page 184.

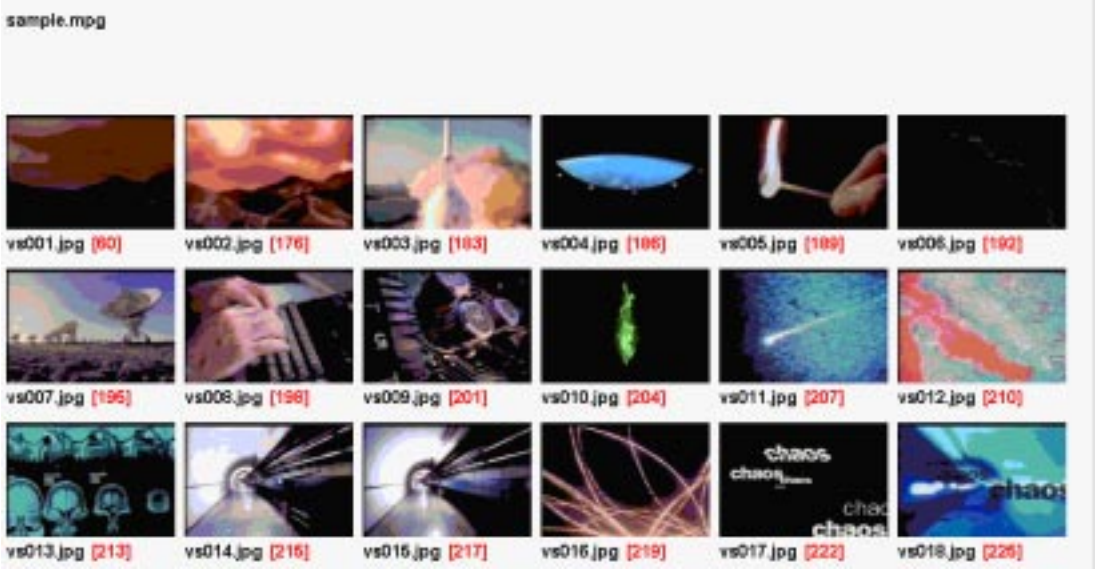


Figure 28. A video storyboard. Representative frames summarize the content and flow of a video.

The studio is planning to use the video scene change detection capabilities of the Video Extender to capture representative frames for their storyboards.

A **video scene change** is a point in a video clip where there is a significant difference between two successive frames. This happens, for example, when the camera recording a video changes its point of view. The frames between two scene changes constitute a **shot**.

When the Video Extender detects a scene change[6], it records data for the associated shot. The data includes the number of the frame that begins the shot, the number of the frame that ends the shot, and the number of a representative frame within the shot. The shot data also includes the pixel content of the representative frame.

Finding and using scene changes

The Video Extender provides a set of application programming interfaces that you can use to find shots or frames in a video clip. After you find the shot or frame, you can access data about it, such as the starting and ending frame numbers of the shot, or the pixel content of a frame. You can then pass this

6. The video scene change detection code includes the University of California at Berkeley MPEG decoder, with modifications from the Boston University Multimedia Communication Laboratory.

information to a program for further processing. For example, you can pass the content of a frame to a program capable of displaying it.

The Video Extender also provides APIs to store shot data in a **shot catalog**. The shot catalog can be in a database or a file. You can access the shot catalog in a file or you can access a read-only view of the shot catalog in a database.

A shot catalog file contains fields for the following data:

- Shot catalog name
- Values that control how the Video Extender detects a shot, for example, the minimum number of frames in a shot
- Values that control how many frames, and which frames, will be stored as representative frames for a shot
- Shot number
- Starting frame number
- Ending frame number
- Representative frame number
- Name of a file that contains the contents of the representative frame

The view of the shot catalog in a database contains columns for the following data:

- Shot handle
- Video table name
- Video column name
- Video handle
- Video file name
- Starting frame number
- Ending frame number
- Representative frame number
- Representative frame data
- Comment

You can access the data in a shot catalog file or query the data when the shot catalog is in a database. The representative frame information is particularly useful in displaying storyboards. In addition, if the shot catalog is in a database, you can join the shot data with related data in other tables. For example, personnel in the video studio can create a shot catalog in a database. They can then join the catalog data with a table that contains the video clips as well as information about the clips. In this way they will be able to use a single query to retrieve a clip and business information about the clip, as well as identify shots within the clip.

Shot detection data structures

Data related to shot detection is stored in structures that are included in the shot detection header file, dmbshot.h. Many of the shot-detection APIs require that you point to one or more of these structures. Some of these structures are used to contain data that the Video Extender uses as input. For example, the shot control structure contains information that controls shot detection. Most of the structures are used by the Video Extender to store data it retrieves from a video. For example, the video frame data structure contains the pixel content of a frame.

The structures used for shot detection are DBvIOType, DBvShotControl, DBvShotType, DBvFrameData, and DBvStoryboardCtrl.

DBvIOType

The DBvIOType data structure contains data about a video, such as its format, dimensions, and number of frames. The data structure is defined as follows:

```
typedef struct {

   FILE *hFile;                    /* file handle for the video */
   char vhandle[255];              /* video handle (if from database)
   char vtable[255];               /* video table name (if from database) */
   char vcolumn[255];              /* video column name (if from database) */
   char vFile[255];                /* name of video file */
   char idxFile[255];              /* name of index file */
   char isIdx;                     /* 1 if the index exists, 0 otherwise */
   char isInDb;                    /* 1 if from DB, 0 if from file */
   int format;                     /* Format of the video */
   unsigned long dx, dy;           /* Dimensions of the video */
   unsigned long totalFrames;      /* TotalFrames in the video */
   unsigned long markFrame;        /* used by shot detection */
   unsigned long currentFrame;     /* The current video frame */
   DBvFrameData fd;                /* Frame data for current frame */
   DBvDCFrameData fdDc;            /* Frame data for DC images */
   unsigned char BGRValid;         /* reserved */
   unsigned short usDeviceID;      /* reserved */
   unsigned long hwnd;             /* reserved */
   int videoReset;                 /* Flag if video is opened or seeked */
   int firstshot;                  /* Used internally to indicate the first call */
   void *reserved                  /* reserved */

} DBvIOType;
```

DBvShotControl

The DBvShotControl data structure contains information used to control shot detection, such as detection method. The data structure is defined as follows:

```
typedef struct {

  unsigned long reserved;
  unsigned long method;           /* detection method */

    #define DETECT_CORRELATION  0x00000001
    #define DETECT_HISTOGRAM    0x00000002
    #define DETECT_CORRHIST     0x00000003
    #define DETECT_CORRHISTDISS 0x00000004


  int normalCorrValue;            /* Correlation threshold */
  int sceneCutSkipXY;             /* reserved */
  int CorrHistThresh;             /* Histogram threshold */
  int DissThresh;                 /* Dissolve threshold */
  int DissCacheSize;              /* Dissolve cache size */
  int DissNumCaches;              /* Dissolve cache number */
  int minShotSize;                /* Minimum frames in a shot */

} DBvShotControl;
```

Table 11 describes each field in DBvShotControl and its allowed and default settings. To initialize these fields to default values, use the DBvInitShotControl API as described in “Initializing values in shot detection data structures” on page 174 .

DBvShotControl settings depend on the type of video: Scene changes in digitized video vary greatly depending on the content and format of the video. Also, the accuracy of the scene change algorithms vary depending on the video. Clearly defined scene changes with obvious differences in overall frame appearance are detected more accurately than more subtle types of changes, or changes where the overall color content remains the same. Although the default DBvShotControl field settings work well for most applications, you might need to adjust these settings to reduce instances of false or missed detection.

Table 11. DBvShotControl fields

Field	Meaning
method	Identifies the method that the Video Extender uses to detect a scene change. You can choose one of the following methods: DETECT_CORRELATION. Compare pixels in two successive frames. If the difference exceeds the correlation threshold, detect a scene change. DETECT_HISTOGRAM. Compare the histogram values of two successive frames. The histogram value measures the distribution of colors in the frame. If the difference exceeds the histogram threshold, detect a scene change. DETECT_CORRHIST. Use the correlation method to identify possible scene changes, then use the histogram method for the frames marked as possible scene changes. If the histogram threshold is exceeded, detect a scene change. DETECT_CORRHISTDISS. Same as for DETECT_CORRHIST, but examine additional frames for dissolves. The default method is DETECT_CORRHIST.
normalCorrValue	An integer value of 0 to 100 that specifies the correlation threshold. This gives the mimimum value of the correlation coefficient between pixels in two frames. A value of 0 means always detect a scene change for the next frame. A value of 100 means detect a scene change only if all the pixels change from one frame to the next frame. The default value is 60.
sceneCutSkipXY	Reserved.
CorrHistThresh	An integer value of 0 to 100 that specifies the histogram threshold. This measures the difference between the histogram values of successive frames. A value of 0 means detect a scene change only if the histogram values are fully different from one frame to the next. A value of 100 means always detect a scene change for the next frame. The default value is 10.
DissThresh	An integer value of 0 to 100 that specifies the dissolve test threshold. This measures the percentage of pixels in a frame that must pass a dissolve test for a dissolve to be detected. A value of 0 means always detect a dissolve for the frame. A value of 100 means detect a dissolve only if all pixels in the frame pass the dissolve test. The default value is 15.
DissCacheSize	An integer value that specifies the number of frames used in the slope portion of the dissolve test. The default value is 4.
DissNumCaches	An integer value that specifies the number of frames used in the consistency portion of the dissolve test. The default value is 7.
minShotSize	An integer value that specifies the minimum number of frames for a shot. For a shot to be detected, it must have at least as many frames as the mimumum. The default value is 5.

DBvShotType

The DBvShotType data structure contains information about a shot, such as its starting frame number, ending frame number, and representative frame number; and a pointer to the pixel content of the representative frame. The data structure is defined as follows:

```
typedef struct {

  unsigned long startFrame;       /* starting frame number */
  unsigned long endFrame;         /* ending frame number */
  unsigned long repFrame;         /* representative frame number */
  DBvFrameData fd;                /* data for representative shot */
  unsigned long dx;               /* frame data width in pixels */
  unsigned long dy;               /* frame data height in pixels */
  char *comment;                  /* shot remark */

} DBvShotType;
```

DBvFrameData

The DBvFrameData data structure contains the pixel content of a frame. The data structure is defined as follows:

```
typedef struct                     /* video frame data */
{
  /* MPEG 1 pixels */
  unsigned char *luminance;        /* Luminance pixel plane (black and white) */
  unsigned char *Cr;               /* Cr pixel plane */
  unsigned char *Cb;               /* Cb pixel plane */
  unsigned char *reserved;

  } DBvFrameData;
```

DBvStoryboardCtrl

The DBvStoryboardCtrl data structure contains values that control which, and how many, representative frames for a shot are stored in a video catalog. See “Building a storyboard” on page 186 for a description of how these values are used. The data structure is defined as follows:

```
typedef struct {

  int thresh1;                    /* threshold for small to medium scenes */
  int thresh2;                    /* threshold for medium to large scenes */
  int delta;                      /* offset used for representative frames */

} DBvStoryboardCtrl;
```

Table 12 describes each field in DBvStoryboardCtrl and its default settings. To initialize these fields to default values, use the DBvInitStoryboardCtrl API as described in “Initializing values in shot detection data structures” on page 174.

DBvStoryboardCtrl settings depend on the type of video: Which, and how many, representative frames are optimal for a storyboard might differ for different types of videos. Although the default DBvStoryboardCtrl field settings work well for many types of videos, you might want to use these settings on a test subset of videos. You can then tune the settings as appropriate before building storyboards for a wider set of videos.

Table 12. DBvStoryboardCtrl fields

Field	Meaning
thresh1	Identifies the threshold for short shots. Shots having fewer frames than the value of thresh1, are short shots. If cataloged, the information for a short will include one representative frame (the middle frame). The default value is 90. If the value of thresh1 is set to -1, a shot will be considered a short shot (regardless of its actual length).
thresh2	Identifies the threshold for medium to large shots. Shots having as many as or fewer frames than the value of thresh2, but as least as many frames as the value of thresh1, are considered medium shots. If cataloged, the information for a medium shot will include two representative frames. The position of the representative frames is controlled by the value of the delta field. Shots having more frames than the value of thresh2, are considered long shots. If cataloged, the information for a long shot will include three representative frames. The position of the first and last representative frames is controlled by the value of the delta field. The second frame is the middle frame. The default value is 150. If the value of thresh2 is set to -1, a shot will be considered a short shot (regardless of its actual length).
delta	Identifies the offset used for representative frames. For medium and long shots, the first representative frame is offset from the beginning of the shot by the number of frames in delta. The last representative frame is offset from the end of the shot by the number of frames in delta. The default value is 5.

Initializing values in shot detection data structures

The values in the DBvShotControl data structure control shot detection. The values in the DBvStoryboardCtrl data structure control the building of a storyboard. You can explicitly specify values for the fields in these data structures. In addition, you can initialize the values in these structures to default values. See Table 11 on page 172 for the default values in the DBvShotControl data structure. See Table 12 for the default values in the DBvStoryboardCtrl data structure.

Use the DBvInitShotControl API to initialize the values in the DBvShotControl data structure. When you use the API, you need to specify the shot control structure. For example, the following statement initializes the fields in the DBvShotControl structure to default values:

```
DBvShotControl    shotCtrl;

rc=DBvInitShotControl(
              shotCtrl);          /* pointer to shot control structure */
```

Use the DBvInitStoryboardCtrl API to initialize the values in the DBvStoryboardCtrl data structure. When you use the API, you need to specify the storyboard control structure. For example, the following statement initializes the fields in the DBvStoryboardCtrl structure to default values:

```
DBvStoryboardCtrl    sbCtrl;

rc=DBvInitStoryboardCtrl(
              sbCtrl);          /* pointer to storyboard control structure */
```

Getting a shot or frame

You can use the Video Extender to get a shot or frame from a video. Before you can get a shot or frame, you must open the video for shot detection. The Video Extender uses an index to access frames and shots. Before you get a shot or frame, you must create an index for the video.

After a video is opened and an index is created, you can get the next shot or frame in a video or get a specific frame by frame number. The Video Extender can process video clips in MPEG-1 format. If you plan to use a retrieved frame with a program that requires RGB format, you can convert the frame to that format using a Video Extender API.

Opening a video for shot detection

Use the DBvOpenFile API to open a video stored in a file. The file must be accessible from the client. Use the DBvOpenHandle API to open a video stored in a database table. The application must first connect to the database. If the video is stored in a database table, the Video Extender copies the video to a temporary file. The temporary file is located in a directory specified in the client environment variable DB2VIDEOTEMP. Opening a video initializes it for shot detection. The Video Extender sets a pointer to the beginning of the video, that is, frame 0.

When you use either API, you need to point to an area used to contain the pointer to the video data structure (DBvIOType). The Video Extender allocates this structure in response to the API call, and uses the structure to store information about the video. The structure also points to the frame data

structure (DBvFrameData) that contains the pixel content of the current frame. For a description of these structures see “Shot detection data structures” on page 170. For the DBvOpenFile API, you also need to specify the name of the video file. For the DBvOpenHandle API, you also need to specify the video handle.

For example, the following statement opens a video for shot detection that is stored in a file:

```
DBvIOType    *videoptr;

rc=DBvOpenFile (
       &videoptr,                        /* pointer to video structure pointer */
       "/employee/video/rsmith.mpg");    /* video file */
```

The following statement opens a video for shot detection that is stored in a database table:

```
EXEC SQL BEGIN DECLARE SECTION;
char Vid_hdl[251];
EXEC SQL END DECLARE SECTION;

DBvIOType    *videoptr;

EXEC SQL SELECT VIDEO INTO :Vid_hdl
FROM EMPLOYEE
WHERE NAME="Anita Jones";

rc=DBvOpenHandle(
       &videoptr,                         /* pointer to video structure pointer */
       vid_hdl);                          /* video handle*/
```

Indexing a video

The Video Extender uses an index to access frames and shots in a video. You need to create an index for a video before you can get a shot or frame from the video (the MPEG format does not provide an index for frames and shots). The index maps frame numbers to the bit streams that comprise an MPEG-1 video.

You can create an index for a video using the DBvCreateIndexFromVideo API or the DBvCreateIndex API. However, if you have opened the video for shot detection using the DBvOpenFile API or the DBvOpenHandle API, you do not have to explicitly create an index; the Video Extender will automatically have created an index for you. (See “Opening a video for shot detection” on page 175 for information about how to open the video)..

When an index is created (either explicitly or automatically), the DB2 Video Extender attempts to store the index in the same path as the video file. It first attempts to store the index file as fname.ext.idx, where fname is the name of

the video file, and ext is the extension of the video file. If that attempt fails, the Video Extender attempts to store the file as fname.idx in the same directory as the video file. If that fails, it attempts to store the index file in the local directory, first as fname.ext.idx, and second as fname.idx.

When the file is opened the Video Extender looks for the index file in the following order:

1. A writeable version of the index file, over a read-only version.
2. An index file in the same path as the video file, over one in the current directory.
3. An index with the name fname.ext.idx, over one with the name fname.idx, where fname is the name of the video file and ext is the extension of the video file.

For example, if an index is created for a video file named myvideo.mpg, the Video Extender will look first for a writeable index named myvideo.mpg.idx in the same path as the video file.

When you use the DBvCreateIndexFromVideo API, specify the DBvIOType data structure. The Video Extender stores the name of the index file in the structure. For a description of this structure, see “Shot detection data structures” on page 170. For example, the following statement creates an index for a video that has previously been opened for shot detection:

```
DBvIOType    *video;

rc=DBvCreateIndexFromVideo(
       video);                  /* pointer to video structure */
```

When you use the DBvCreateIndex API, specify the name of the video file. The Video Extender stores the index in a file (in the same directory in which the video resides). For example, the following statement creates an index for a video file (the file was not previously opened for shot detection):

```
rc=DBvCreateIndex(
       "/employee/video/rsmith.mpg");    /* video file */
```

You can also determine whether an index exists for a video. Use the DBvIsIndex API to check for an index. The API sets a status variable to 0 if no index exists for the video, or 1 if an index exists for the video. For example, the following statement checks for the existence of an index for a video file:

```
short  *status

rc=DBvIsIndex(
       "/employee/video/rsmith.mpg",     /* video file */
       &status);                         /* status indicator */
```

Back up the video index: Back up the video index file in case you need to recover it. The file is located in the directory where the Video Extender is installed.

Getting a frame

You can get the current frame in a video. You an also set the current frame to be a particular frame number. Use the DBvGetFrame API to get the current frame in a video. Use the DBvSetFrameNumber API to set the current frame to a particular frame number.

When you use the DBvGetFrame API, specify the video structure. For example, the following statement gets the current frame in a video:

```
DBvIOType    *video;

rc=DBvGetFrame(
       video);                  /* pointer to video structure */
```

When you use the DBvSetFrameNumber API, specify the video structure and the number of the frame you want set as the current frame. For example, the following statements set the current frame to frame number 85 and then get the frame:

```
DBvIOType    *video;

rc=DBvSetFrameNumber(
       video,                   /* pointer to video structure */
       85);                     /* frame number */

rc=DBvGetFrame(
       video);                  /* pointer to video structure */
```

On output, the DBvSetFrameNumber API resets the currentFrame field in the DBvIOType structure. The DBvGetFrame API puts the pixel content of the frame in the DBvFrameData structure. For a description of these structures, see “Shot detection data structures” on page 170.

Getting a shot

Use the DBvDetectShot API to get the next shot in a video. When you use the DBvDetectShot API, you need to point to the following data structures:

- Video (DBvIOType)
- Shot control (DBvShotControl)
- Shot type (DBvShotType)

You also need to point to a starting frame for the search. The Video Extender will begin its search for the next shot from that point in the video.

As output from the API, the Video Extender sets a shotDetected flag and points to the starting frame of the next shot and its frame data. If the shotDetected flag is set to 1, a shot has been detected. In this case, the Video Extender:

- Sets the currentFrame field in DBvIOType to the starting frame of the next shot
- Puts the data for the starting frame of the next shot in the fd field in DBvIOType
- Sets DBvShotType to contain the starting frame number, ending frame number, representative frame number, representative frame data, and comment for the next shot

If the shotDetected flag is set to 0, a shot has not been detected. In this case, the Video Extender returns a code indicating that the end of the video was reached.

For a description of these structures, see “Shot detection data structures” on page 170.

The following statements, for example, requests the next shot in a video:

```
DBvIOType    *video;
long start_frame = 1;
char shotDetected = 0;
DBvShotControl    shotCtrl;
DBvShotType    shot;

shotCtrl→method=DETECT_CORRHIST
shotCtrl→normalCorrValue=60;
shotCtrl→sceneCutSkipXY=1;
shotCtrl→CorrHistThresh=10;
shotCtrl→DissThresh=10;
shotCtrl→DissCacheSize=4;
shotCtrl→DissNumCaches=7;
shotCtrl→minShotSize=0;

rc=DBvDetectShot(
       video,              /* pointer to video structure */
       start_frame,        /* starting frame for search */
       &shotDetected,      /* shot detected flag */
                           /* 1=detected, 0=not detected */
       shotCtrl,           /* pointer to shot control structure */
       &shot);             /* pointer to shot type structure */
```

Converting the format of a retrieved frame

The content of an MPEG-1 frame is in YUV format, a format that includes information about the luminance pixel plane, Cr pixel plane, and Cb pixel plane of a frame. If you want to edit the video frame, you may find it

convenient to convert the frame from YUV to RGB format. The Video Extender provides the DBvFrameDatato24BitRGB API to convert a retrieved MPEG-1 frame from YUV format to 24-bit RGB format. To use the API, you must first allocate a target buffer.

When you issue the API, you need to point to the target buffer and the frame data that you want to convert. You also need to specify the height and width of the frame. (You can get the data, height, and width of the frame from the DBvIOType structure for the frame.) For example, the following statements convert an MPEG-1 frame to 24-bit RGB format:

```
char RGB[18000];
DBvIOType      *video;
DBvFrameData   fd;

rc=DBvGetNextFrame(
       video);                  /* pointer to video structure */

fd=video.fd
dx=video.dx
dy=video.dy

rc=DBvFrameDataTo24BitRGB (
       RGB,                      /* pointer to target buffer */
       &fd,                      /* pointer to frame data */
       dx,                       /* frame width */
       dy);                      /* frame height */
```

Closing a video file

Use the DBvClose API to close a video file that has been opened for shot detection. When you use the API, you specify a pointer to the video structure for the file.

For example, the following statement closes a video file that has been open for shot detection:

```
DBvIOType    *video;

rc=DBvClose (video);
```

Displaying a retrieved frame

The content of a retrieved MPEG-1 frame is in YUV format; this is not a format that is displayable by most image display programs. To display a retrieved video frame, you need to put it in a format that an image display program can understand, such as BMP format. For example, to display an MPEG-1 frame:

1. Use the DBvFrameDatato24BitRGB API to convert the format of a retrieved MPEG-1 frame from YUV format to 24-bit RGB format. See

"Converting the format of a retrieved frame" on page 179 for information on using the DBvFrameDatato24BitRGB API.

2. Append an appropriate header to the converted frame. For example, BMP format requires a header that includes information such as the width and height of the image.
3. Copy the frame contents (with its header) to a file.
4. Use the DBiBrowse API to display the file. For information on using the DBiBrowse API, see "Using the display or play APIs" on page 119.

Cataloging shots

You can store information about a shot in a shot catalog. The Video Extender provides APIs to:

- Create and manage a shot catalog in a database. You can use the APIs to:
  - Create a shot catalog in a database
  - Store in a shot catalog information about a single shot
  - Store in a shot catalog information about all the shots in a video
  - Change the information stored for a shot in a shot catalog
  - Merge shot information in a shot catalog
  - Delete shot information from a shot catalog
  - Delete a shot catalog from a database
- Create a shot catalog file and store in it information about all the shots in a video. An API is provided that creates the catalog file and fills it with shot data. You can access and manipulate the data in the shot catalog file, but no APIs are provided to do that.

Cataloged shots provide input for storyboards: After you store shot information in a shot catalog (whether the catalog is in a database or a file), you can use that information in a shot-related application. For example, you can create an application that gets the representative frames for all the shots in a video and displays them in a storyboard.

You only need to create a shot catalog for a database: You need to create a shot catalog only if you want the catalog to reside in a database. The Video Extender automatically creates a shot catalog file when you store data for the shots in a video and indicate that you want the output in a file.

Before you create a catalog (database only)

Before you create and use a catalog in a database, you must:

- Issue an SQLConnect call. A shot catalog in a DB2 database consists of a collection of tables. Before you create a shot catalog in a database or

perform operations on it, you must connect to the database with an SQLConnect call. (SQLConnect is a DB2 Call Level Interface call.) The call returns a connection handle that you need to specify in the APIs that manage the shot catalog.

- Enable the database for image data. You must enable a database for the DB2Image data type before you create a shot catalog in the database. In addition to other information it stores in the shot catalog, the Video Extender stores representative frame data for each cataloged shot. The data type of the representative frame data is DB2Image.

Creating a shot catalog (database only)

Use the DBvCreateShotCatalog API to create a shot catalog in a database. (The Video Extender automatically creates a shot catalog file when you store data for the shots and indicate that you want the output in a file.). The catalog consists of tables that store shot-related information. You can query a view of the tables using SQL. Table 13 shows the columns in the view.

Table 13. Columns in the shot catalog view

Column name	Data type	Description
SHOTHANDLE	CHAR(36)	Shot handle
VIDEOHANDLE	VARCHAR(254)	Video handle. This column contains a value only if the video is opened with the DBvOpenHandle API.
VIDEOTABLE	VARCHAR(254)	Table that contains the video. This column contains a value only if the video is opened with the DBvOpenHandle API.
VIDEOCOLUMN	VARCHAR(254)	Table column that contains the video. This column contains a value only if the video is opened with the DBvOpenHandle API.
VIDEOFILE	VARCHAR(254)	Video file name. This column contains a value only if the video is opened with the DBvOpenFile API.
STARTFRAME	INTEGER	Starting frame number
ENDFRAME	INTEGER	Ending frame number
REPFRAME	INTEGER	Representative frame number
REPFRAMEDATA	DB2IMAGE	Representative frame data

Table 13. Columns in the shot catalog view (continued)

Column name	Data type	Description
COMMENTS	LONG VARCHAR	Comment

You have flexibility in how many shot catalogs you create in a database and for which shots you store information in each shot catalog. You can create one catalog to store shot information for many videos, have shot information for each video stored in a separate catalog, or store information for multiple shots within a video in multiple catalogs.

When you use the API, you need to specify a name for the catalog. Names larger than 16 characters are truncated. You also need to specify the database connection handle returned by the SQLConnect call to the database. For example, the following statements create a shot catalog named hotshots:

```
SQLHDBC  hdbc;

rc = SQLConnect(hdbc,"hotshots",SQL_NTS,id,SQL_NTS,password,SQL_NTS);

rc=DBvCreateShotCatalog(
                "hotshots",          /* shot catalog name */
                hdbc);               /* database connection handle */
```

Shot catalog views are named MMDBSYS.SV*catname*, where *catname* is the name of the shot catalog. For example, a view of the catalog named hotshots is named MMDBSYS.SVHOTSHOTS.

Storing information about a single shot (database only)

Use the DBvInsertShot API to store information about a single shot in a shot catalog. You can store information for a single shot in a video only if the shot catalog is in a database. The information stored in the catalog includes:

- Shot handle
- Video table name (for video clips stored in a table)
- Video column name (for video clips stored in a table)
- Video handle (for video clips stored in a a table)
- Video file name (for video clips stored in a file)
- Starting frame number
- Ending frame number
- Representative frame number
- Representative frame data

A comment, however, is not stored for the shot. See “Specifying a comment for a shot (database only)” on page 188 for a description of how you can add a comment to the information stored for a shot.

Using Scene Changes

When you use the DBvInsertShot API, you need to specify the shot catalog name and a pointer to the shot. One way to set the pointer to the shot is to get the next shot as described in “Getting a shot” on page 178. You also need to specify the database connection handle returned by the SQLConnect call to the database. For example, the following statements get the next shot after frame 1 and then store information about the shot in a shot catalog named hotshots:

```
SQLHDBC  hdbc;
SQLHENV  henv;
DBvIOType    *video;
long start_frame = 1;
char shotDetected = 0;
DBvShotControl    shotCtrl;
DBvShotType    shot;

shotCtrl→method=DETECT_CORRHIST
shotCtrl→normalCorrValue=60;
shotCtrl→sceneCutSkipXY=1;
shotCtrl→CorrHistThresh=10;
shotCtrl→DissThresh=10;
shotCtrl→DissCacheSize=4;
shotCtrl→DissNumCaches=7;
shotCtrl→minShotSize=0;

SQLAllocConnect(henv,&hdbc)

rc = SQLConnect(hdbc,"hotshots",SQL_NTS,id,SQL_NTS,password,SQL_NTS);

rc=DBvDetectShot(
       video,
       start_frame,
       &shotDetected,
       &shotCtrl,
       &shot)

rc=DBvInsertShot (
       "hotshots",              /*shot catalog name*/
       shot,                      /*pointer to shot*/
       hdbc);                     /*database connection handle*/
```

Storing information about all the shots in a video

Use the DBvBuildStoryboardTable API or the DBvBuildStoryboardFile API to store in a shot catalog information for all the shots in a video. The DBvBuildStoryboardTable API stores the information in a shot catalog that resides in a database. The DBvBuildStoryboardFile API creates a shot catalog file and stores the shot information in the file.

For either API, the source video can be in a database table or in a file.

When you use either API, you need to

- Specify the shot catalog name.
- Point to the video structure.
- Point to the DBvShotControl data structure.
- Point to the DBvStoryboardCtrl data structure. Values in this data structure control which, and how many, video frames will be stored as representative frames in the shot catalog. See “Building a storyboard” on page 186 for more information about setting these values.

For the DBvBuildStoryboardTable API only, you also need to specify the database connection handle returned by the SQLConnect call to the database.

For example, the following statements store in a shot catalog information for all the shots in a video. The shot catalog is in a database.

```
SQLHDBC  hdbc;
SQLHENV  henv;
DBvIOType    *video;
DBvShotControl    shotCtrl;
DBvStoryBoardCtrl    sbCtrl;

sbCtrl→thresh1=50
sbCtrl→thresh2=500;
sbCtrl→delta=20;

SQLAllocConnect(henv,&hdbc)

rc = SQLConnect(hdbc,"hotshots",SQL_NTS,id,SQL_NTS,password,SQL_NTS);

rc=DBvBuildStoryboardTable (
       "hotshots",               /*shot catalog name*/
       video,                    /*pointer to video structure*/
       shotCtrl,                 /*pointer to shot control structure*/
       sbctrl,                   /*pointer to storyboard control structure*/
       hdbc);                    /*database connection handle*/
```

The following statements create a shot catalog file and store in the file information for all the shots in a video.

```
DBvIOType    *video;
DBvShotControl    shotCtrl;
DBvStoryBoardCtrl    sbCtrl;

sbCtrl→thresh1=50
sbCtrl→thresh2=500;
sbCtrl→delta=20;

rc=DBvBuildStoryboardFile (
       "hotshots",               /*shot catalog file name*/
       video,                    /*pointer to video structure*/
       shotCtrl,                 /*pointer to shot control structure*/
       sbctrl);                  /*pointer to storyboard control structure*/
```

Building a storyboard

As their name implies, the DBvBuildStoryboardTable API and DBvBuildStoryboardFile API are particularly useful for storing information to be used in a storyboard. A **storyboard** is a visual summary of a video. You can create a storyboard by displaying the representative frames stored for a video in a shot catalog.

The DBvBuildStoryboardTable API and DBvBuildStoryboardFile API store one or more representative frames for a shot. Values that you specify in the DBvStoryboardCtrl structure control the number of representative frames stored for a shot and which frames will be used. See "Shot detection data structures" on page 170 for the definition of the DBvStoryboardCtrl structure. Figure 29 illustrates how the values in the DBvStoryboardCtrl fields are used.

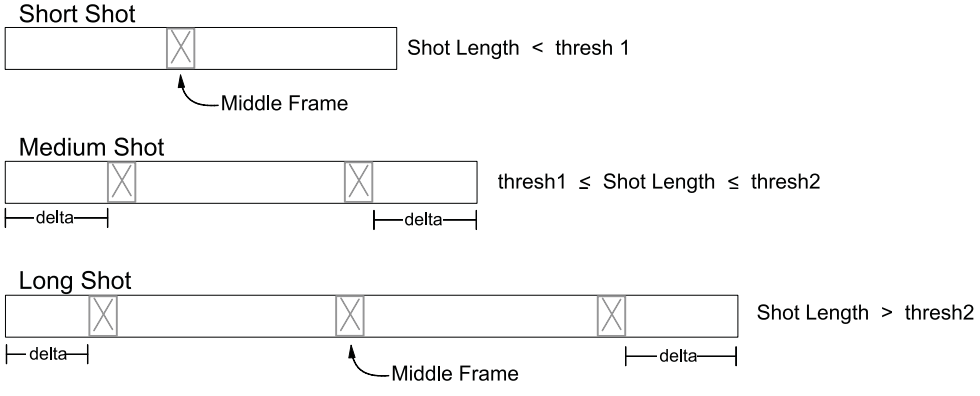


Figure 29. How values in the DBvStoryboardCtrl structure are used

As Figure 29 illustrates:

- Only one representative frame is stored for a short shot. The number of frames in a short shot is less than the thresh1 value in the DBvStoryboardCtrl data structure. The representative frame is the middle frame of the shot.
- Two representative frames are stored for a medium shot. The number of frames in a medium shot is greater than or equal to the thresh1 value and less than or equal to the thresh2 value in the DBvStoryboardCtrl data structure. The delta value in the DBvStoryboardCtrl data structure determines the number of frames from the start of the video to the first representative frame. The delta value also determines the number of frames from the second representative frame to the end of the video.
- Three representative frames are stored for a long shot. The number of frames in a long shot is greater than the thresh2 value in the

DBvStoryboardCtrl data structure. The delta value in the DBvStoryBoardCtrl data structure determines the number of frames from the start of the video to the first representative frame. The second representative frame is the middle frame of the video. The distance of the third representative frame from the end of the video is determined by the delta value.

Any shot can be processed as a short shot if the thresh1 or thresh2 value is set to -1. In this case, only one representative frame, the middle frame, will be stored for the shot in the shot catalog.

In addition to the values in the DBvStoryboardCtrl data structure, a number of fields in the DBvShotControl data structure impact which representative frames are stored for subsequent display in a storyboard. For example, the CorrHistThresh, normalcorrValue, and minShotSize fields in the DBvShotControl data structure specify thresholds for shot detection, and thus affect what frames will be displayed in a storyboard of a video. When you use the DBvBuildStoryBoardTable API and DBvBuildStoryBoardFile API to store shot information for use in a storyboard, you might first do a trial run using initial settings for the DBvStoryBoardCtrl and DBvShotControl data structures. You can then tune your results by changing the values in various fields of these data structures.

Displaying a storyboard

You can create a program to display a storyboard. You do this by accessing the representative frames stored in a shot catalog for a video. If the DBvBuildStoryboardFile API was used to store the shots for the video, the shot catalog file points to GIF files for the representative frames. You can display these GIF files using a browser or display program as appropriate.

If the DBvBuildStorybBoardTable API was used to store the shots for the video, the shot catalog (stored in a database) contains data for the representative frames. You can access the representative frame data in the shot catalog view (see Table 13 on page 182 for a description of the view). The representative frame data is in YUV format; this is not a format that is displayable by most image display programs. To display the representative frames, you can convert the frame data using the DBvFrameDatato24BitRGB API as decribed in “Displaying a retrieved frame” on page 180. You can then display the representative frames using a browser or display program as appropriate.

Storyboard sample programs

The SAMPLES subdirectory contains two sample programs that demonstrate building and displaying a storyboard for a video. One sample program, in file

MAKESF.EXE, uses the DBvBuildStoryBoardFile API to create a shot catalog file and store shot data in the file. The other sample program, MAKEHTML.EXE, accesses the shot catalog file and creates HTML pages for display by a Web browser.

Specifying a comment for a shot (database only)

You can specify a comment to be stored with the other information for a shot in a shot catalog. Use the DBvSetShotComment API to specify the comment.

When you use the API, you need to specify the name of the shot catalog to store the comment, the handle of the shot for which the comment is being added, and the comment. You also need to specify the database connection handle returned by the SQLConnect call to the database. For example, the following statements add a comment for a shot (which starts at frame number 85) in a shot catalog named hotshots:

```
SQLHDBC  hdbc;
SQLHENV  henv;
char shothandle[37];

SQLAllocConnect(henv,&hdbc)

rc = SQLConnect(hdbc,"hotshots",SQL_NTS,id,SQL_NTS,password,SQL_NTS);

EXEC SQL SELECT SHOTHANDLE INTO :shothandle
  FROM MMDBSYS.SVHOTSHOTS
  WHERE STARTFRAME=85;

rc=DBvSetShotComment (
        "hotshots",                     /*shot catalog name*/
        shothandle,                     /*shot handle*/
        "shot of beach at sunset",      /*comment*/
        hdbc);                          /*database connection handle*/
```

Changing the information stored for a shot (database only)

You can change the information that is stored for a shot in a shot catalog. To do that, use the DBvUpdateShot API. Put the replacing information in a DBvShotType structure. You also need to specify information for the remaining fields, even if they are unchanged. When you use the DBvUpdateShot API, specify the name of the catalog and a pointer to the DBvShotType structure. You also need to specify the database connection handle returned by the SQLConnect call to the database.

When you change the information for the shot, you have the option of changing the comment (if any) stored with the information. If you want to change the comment, specify it in the DBvShotType structure. If you want to keep the old comment, specify a null value in the DBvShotType structure.

For example, the following statements change the information stored for a shot in a catalog named hotshots; the shot begins at frame number 85:

```
SQLHDBC  hdbc;
SQLHENV  henv;
char shothandle[37];
DBvShotType    shot;
DBvFrameData  fd110;

/* get shot handle */

EXEC SQL SELECT SHOTHANDLE INTO :shothandle
  FROM MMDBSYS.SVHOTSHOTS
  WHERE STARTFRAME=85;

/* change shot attribute */

shot.startFrame=110;
shot.endFrame=200;
shot.repframe=110;
shot.fd=fd110;
shot.comment=NULL;

/* update shot information */

SQLAllocConnect(henv,&hdbc)

rc = SQLConnect(hdbc,"hotshots",SQL_NTS,id,SQL_NTS,password,SQL_NTS);

rc=DBvUpdateShot (
        "hotshots",                    /*shot catalog name*/
        shot,                          /*shot information*/
        hdbc);                         /*database connection handle*/
```

Merging shot information in a shot catalog (database only)

You can merge the information stored for two shots in a shot catalog. When you merge shot information, you indicate an order for the merge by identifying a first shot and a second shot. The starting frame number of the first shot is stored as the starting frame number of the merged shot. The number of the largest frame between the first and second shots is stored as the ending frame number of the merged shot. A merge replaces the information stored for the first shot with the information for the merged shot; the information stored for the second shot is deleted from the shot catalog.

Use the DBvMergeShots API to merge the information for two shots in a shot catalog. When you use the API, specify the shot catalog name followed by the handle of the first and second shots to be merged. You also need to specify the database connection handle returned by the SQLConnect call to the database. For example, the following statements merge information stored for two shots in a catalog named hotshots; the first shot begins at frame 85, the second shot begins at frame 210:

```
SQLHDBC  hdbc;
SQLHENV  henv;
char shothandle1[37];
char shothandle2[37];

EXEC SQL SELECT SHOTHANDLE INTO :shothandle1
  FROM MMDBSYS.SVHOTSHOTS1
  WHERE STARTFRAME=85;

EXEC SQL SELECT SHOTHANDLE INTO :shothandle2
  FROM MMDBSYS.SVHOTSHOTS2
  WHERE STARTFRAME=210;

SQLAllocConnect(henv,&hdbc)

rc = SQLConnect(hdbc,"hotshots",SQL_NTS,id,SQL_NTS,password,SQL_NTS);

rc=DBvMergeShots (
        "hotshots",                  /*shot catalog name*/
        shothandle1,                 /*shot handle for first shot*/
        shothandle2,                 /*shot handle for second shot*/
        hdbc);                       /*database connection handle*/
```

Deleting shot information from a shot catalog (database only)

Use the DBvDeleteShot API to delete information about a shot from a shot catalog. When you use the API, specify the shot catalog name followed by the shot handle. You also need to specify the database connection handle returned by the SQLConnect call to the database. For example, the following statements delete information about a shot (which begins at frame number 85) in a shot catalog named hotshots:

```
SQLHDBC  hdbc;
SQLHENV  henv;
char shothandle[37];

EXEC SQL SELECT shothandle INTO :shothandle
  FROM mmdbsys.svhotshots
  WHERE startframe=85;

rc=DBvDeleteShot (
        "hotshots",                  /*shot catalog name*/
        shothandle,                  /*shot handle*/
        hdbc);                       /*database connection handle*/
```

Deleting a shot catalog (database only)

Use the DBvDeleteShotCatalog API to delete a shot catalog. When you use the API, specify the name of the shot catalog to be deleted and the database connection handle returned by the SQLConnect call to the database. For example, the following statement deletes a shot catalog named hotshots:

```
SQLHDBC  hdbc;
SQLHENV  henv;

rc=DBvDeleteShotCatalog (
        "hotshots",             /*shot catalog name*/
        hdbc);                  /*database connection handle*/
```

Using Scene Changes

Part 4. Reference

Chapter 15. Distinct Data Types and User-Defined Functions

This chapter gives reference information for the distinct data types and UDFs created by the DB2 extenders.

Distinct Data Types

Table 14 lists and describes the distinct data types created by the DB2 extenders. It also lists the DB2 source data type for each distinct data type.

Table 14. Distinct data types created by the DB2 extenders

Distinct type	Source data type	Description
DB2IMAGE	VARCHAR(250)	Image handle. A variable-length string that contains information needed to access an image object. Image handles are stored in a user table column enabled for the Image Extender.
DB2AUDIO	VARCHAR(250)	Audio handle. A variable-length string that contains information needed to access an audio object. Audio handles are stored in a user table column enabled for the Audio Extender.
DB2VIDEO	VARCHAR(250)	Video handle. A variable-length string that contains information needed to access a video object. Video handles are stored in a user table column enabled for the Video Extender.

User-Defined Functions

This section gives reference information for the DB2 extenders. The UDFs are listed in alphabetical order.

The following information is given for each UDF:

- The extenders that implement the UDF
- A brief description
- The include (header) file for the UDF
- The SQL syntax of the UDF
- A description, including the data type, of the UDF parameters

- The value returned by the UDF, including its data type
- Examples of use

Table 15 lists the UDFs and identifies the extenders that implement each UDF. The table also points to where you can find more information about each UDF. The UDFs in this table can be coded in embedded SQL statements or in DB2 CLI calls.

Table 15. DB2 Extender UDFs

UDF	Description	Image	Audio	Video	See page
AlignValue	Returns the number of bytes per sample in a WAVE audio, or in an audio track of a video.		x	x	201
AspectRatio	Returns the aspect ratio of the first track of an MPEG1 and MPEG2 video.			x	203
BitsPerSample	Returns the number of bits of data used to represent each sample of WAVE or AIFF audio in an audio, or in an audio track of a video.		x	x	204
BytesPerSec	Returns the data transfer rate, in average bytes per second, for a WAVE audio.		x		205
Comment	Returns or updates a comment stored with an image, audio, or video.	x	x	x	206
CompressType	Returns the compression format, such as MPEG-1, of a video.			x	208
Content	Retrieves or updates the content of an image, audio, or video from a database.	x	x	x	209
DB2Audio	Stores the content of an audio in a database table.		x		215
DB2Image	Stores the content of an image in a database table.	x			219
DB2Video	Stores the content of a video in a database table.			x	224
Duration	Returns the duration (that is, playing time in seconds) of a WAVE or AIFF audio, or video.		x	x	228

Table 15. DB2 Extender UDFs (continued)

UDF	Description	Image	Audio	Video	See page
Filename	Returns the name of the server file that contains the contents of an image, audio, or video.	x	x	x	229
FindInstrument	Returns the track number of the first occurrence of a specified instrument in a MIDI audio.		x		230
FindTrackName	Returns the number of a specified named track in a MIDI audio.		x		231
Format	Returns the format of an image, audio, or video.	x	x	x	232
FrameRate	Returns the throughput of a video in frames per second.			x	233
GetInstruments	Returns the instrument name of all instruments in a MIDI audio.		x		234
GetTrackNames	Returns the name of all tracks in a MIDI audio.		x		235
Height	Returns the height, in pixels, of an image or video frame.	x		x	236
Importer	Returns the user ID of the person who stored an image, audio, or video in a database table.	x	x	x	237
ImportTime	Returns a timestamp that indicates when an image, audio, or video was stored in a database table.	x	x	x	238
MaxBytesPerSec	Returns the maximum throughput of a video in bytes per second.			x	239
NumAudioTracks	Returns the number of audio tracks in a video or MIDI audio.		x	x	240
NumChannels	Returns the number of recorded audio channels in a WAVE or AIFF audio, or video.		x	x	241
NumColors	Returns the number of colors in an image.	x			242
NumFrames	Returns the number of frames in a video.			x	243
NumVideoTracks	Returns the number of video tracks in a video.			x	244

Table 15. DB2 Extender UDFs (continued)

UDF	Description	Image	Audio	Video	See page
QbScoreFromName	Returns the score of an image (uses a named query object). (Replaces QbScore.)	x			245
QbScoreFromString	Returns the score of an image (uses a query string).	x			247
QbScoreTBFromName	Returns a table of scores from an image column (uses a named query object).	x			248
QbScoreTBFromStr	Returns a table of scores from an image column (uses a query string).	x			250
Replace	Updates the content of an image, audio, or video stored in a database, and updates its comment.	x	x	x	252
SamplingRate	Returns the sampling rate of a WAVE or AIFF audio, or of an audio track in a video, in number of samples per second.		x	x	256
Size	Returns the size of an image, audio, or video, in bytes.	x	x	x	257
Thumbnail	Returns or updates a thumbnail-size version of an image or video frame stored in a database.	x		x	258
TicksPerQNote	Returns the clock speed of a recorded MIDI audio, in ticks per quarter note.		x		260
TicksPerSec	Returns the clock speed of a recorded MIDI audio, in ticks per second.		x		261
Updater	Returns the user ID of the person who last updated an image, audio, or video in a database table.	x	x	x	262
UpdateTime	Returns a timestamp that indicates when an image, audio, or video in a database table was last updated.	x	x	x	263
Width	Returns the width in pixels of an image or video frame.	x		x	264

AlignValue

Image	Audio	Video
	X	X

Returns the number of bytes per sample in a WAVE audio, or in an audio track of a video. A WAVE audio can store its data using one byte per sample (8-bit mono, referred to as “byte aligned”), two bytes per sample (8-bit stereo or 16-bit mono, referred to as “word aligned”), or four bytes per sample (16-bit stereo, referred to as “double-word aligned”).

Include file

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2AUDIO or DB2VIDEO)
Column name or host variable that contains the handle of the audio.

Return values (data type)

Bytes per sample value of WAVE audio, or audio track in a video (SMALLINT). Values can be:

1 byte aligned

2 word aligned

4 double-word aligned

Null value audio in other formats

Examples

Get the file name of all audios stored in the sound column of the employee table that are word aligned:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_fname[251];
EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL SELECT FILENAME(SOUND)
      INTO :hvAud_fname
      FROM EMPLOYEE
      WHERE ALIGNVALUE(SOUND) = 2;
```

Find the bytes per sample value of an audio track in a video; the video is stored in the video column of the employee table for Anita Jones:

```
EXEC SQL BEGIN DECLARE SECTION;
      short hvAlign_val;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT ALIGNVALUE(VIDEO)
      INTO :hvAlign_val
      FROM EMPLOYEE
      WHERE NAME='Anita Jones';
```

AspectRatio

Image	Audio	Video
		X

Returns the aspect ratio of the first track of an MPEG video.

Include file

dmbvideo.h

Syntax

Parameters (data type)

handle (DB2VIDEO)
Column name or host variable that contains the handle of the video.

Return values (data type)

Aspect ratio of first track of MPEG video, or a null value for video in other formats (SMALLINT)

Examples

Get the aspect ratio of the video stored for Robert Smith in the video column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
      short hvAsp_ratio;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT ASPECTRATIO(VIDEO)
      INTO :hvAsp_ratio
      FROM EMPLOYEE
      WHERE NAME='Robert Smith';
```

BitsPerSample

Image	Audio	Video
	X	X

Returns the number of bits of data used to represent each sample of WAVE or AIFF audio in an audio, or in an audio track of a video.

Include file

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2AUDIO or DB2VIDEO)
Column name or host variable that contains the handle of the audio or video.

Return values (data type)

Number of bits of data used to represent each sample of video or WAVE or AIFF audio (SMALLINT). Returns a null value for audio in other formats

Examples

Get the file name of all WAVE audios stored in the sound column of the employee table whose bits per sample is equal to 8:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
      INTO :hvAud_fname
      FROM EMPLOYEE
      WHERE FORMAT(SOUND)='WAVE'
      AND BITSPERSAMPLE(SOUND) = 8;
```

BytesPerSec

Image	Audio	Video
	X	

Returns the data transfer rate, in average bytes per second, for a WAVE audio.

Include file

dmbaudio.h

Syntax

Parameters (data type)

handle (DB2AUDIO)
Column name or host variable that contains the handle of the audio.

Return values (data type)

Data transfer rate (INTEGER). Returns a null value for audio in other formats.

Examples

Get the file name of all audios stored in the sound column of the employee table whose transfer rate, in average bytes per second, is less than 44100:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
      INTO :hvAud_fname
      FROM EMPLOYEE
      WHERE BYTESPERSEC(SOUND) < 44100;
```

Comment

Image	Audio	Video
X	X	X

Returns or updates a comment stored with an image, audio, or video.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Retrieve comment

Syntax

Update comment

Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

new_comment (LONG VARCHAR)
New comment for update. A null value or empty string deletes the existing comment.

Return values (data type)

For update, the handle of the image, audio, or video (DB2IMAGE, DB2AUDIO, or DB2VIDEO). For retrieval, the comment (LONG VARCHAR).

Examples

Get the file name of all images from the picture column of the employee table that have the word “confidential” in associated comments:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvImg_fname[255;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE COMMENT(PICTURE)
        LIKE '%confidential%';
```

Update the comment associated with the Anita Jones's video clip in the video column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
struct{
      short len;
      char data[4000];
      }hvRemarks;
EXEC SQL END DECLARE SECTION;

/* Get the old comment */

EXEC SQL SELECT COMMENT(VIDEO)
     INTO :hvRemarks
     FROM EMPLOYEE
     WHERE NAME =  'Anita Jones';

/* Update the comment */

hvRemarks.data[hvRemarks.len]='\0';
strcat (hvRemarks.data, "Updated video");
hvRemarks.len=strlen(hvRemarks.data);

EXEC SQL UPDATE EMPLOYEE
     SET VIDEO=COMMENT(VIDEO, :hvRemarks)
     WHERE NAME =  'Anita Jones';
```

CompressType

Image	Audio	Video
		X

Returns the compression format, such as MPEG-1, of a video.

Include file

dmbvideo.h

Syntax

Parameters (data type)

handle (DB2VIDEO)
Column name or host variable which contains the handle of the video

Return values (data type)

Compression format of the video (VARCHAR(8))

Examples

Get the names of all videos stored in the video column of the employee table whose compression format is MPEG-1:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvVid_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(VIDEO)
     INTO :hvVid_fname
     FROM EMPLOYEE
     WHERE COMPRESSTYPE(VIDEO) = 'MPEG1';
```

Content

Image	Audio	Video
X	X	X

Retrieves or updates the content of an image, audio, or video from a database. The content can be retrieved to a client buffer, client file, or server file.

Include file

image dmbimage.h

audio dmbaudio.h

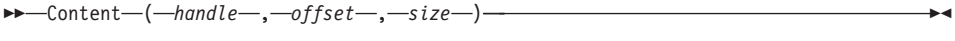
video dmbvideo.h

Syntax

Retrieve content to buffer or client file

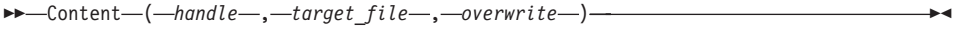
Syntax

Retrieve a segment of content to buffer or client file



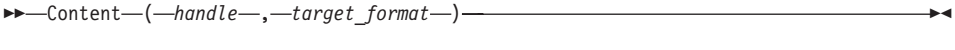
Syntax

Retrieve content to server file



Syntax

Retrieve content to buffer or client file with format conversion—image only



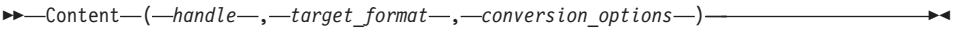
Content

Syntax

Retrieve content to server file with format conversion—image only

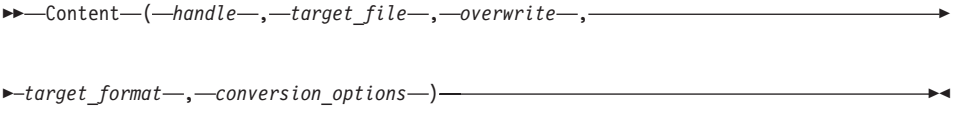
Syntax

Retrieve content to buffer or client file with format conversion and additional changes—image only



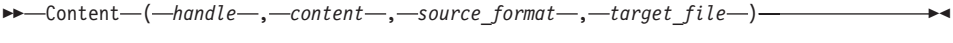
Syntax

Retrieve content to server file with format conversion and additional changes—image only



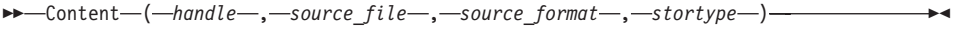
Syntax

Update content from buffer or client file



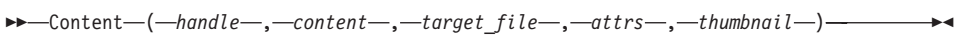
Syntax

Update content from server file



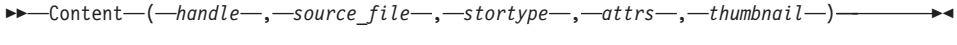
Syntax

Update content with user-supplied attributes from buffer or client file



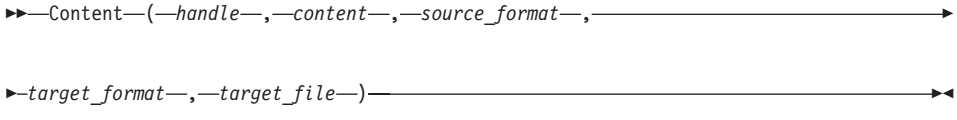
Syntax

Update content with user-supplied attributes from server file



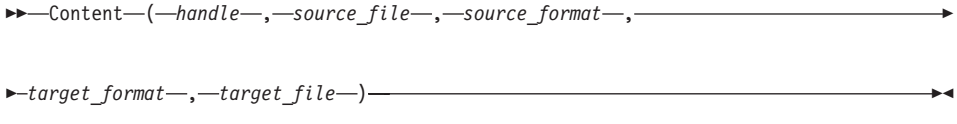
Syntax

Update content from buffer or client file with format conversion—image only



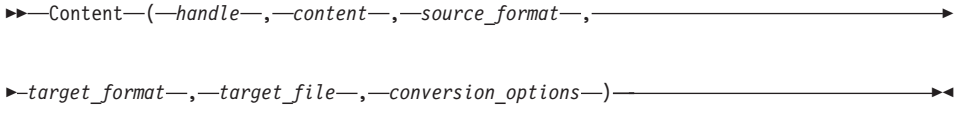
Syntax

Update content from server file with format conversion—image only



Syntax

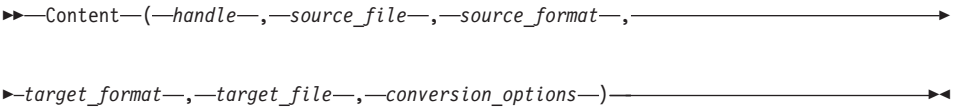
Update content from buffer or client file with format conversion and additional changes—image only



Content

Syntax

Update content from server file with format conversion and additional changes—image only



Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

offset (INTEGER)
Starting offset (origin 1) of an image, audio, or video to be retrieved.

size (INTEGER)
Number of bytes of an image, audio, or video to be retrieved.

source_file (LONG VARCHAR)
The name of the file that contains the content for update of the image, audio, or video.

target_file (LONG VARCHAR)
For retrieval, the name of the file into which the image, audio, or video is to be retrieved. For update, the name of the file that contains the image, audio, or video to be updated.

stortype (INTEGER)
A value that indicates where the updated image, audio, or video will be stored. The constant MMDB_STORAGE_TYPE_INTERNAL (value=1) indicates that the updated object will be stored in the database as a BLOB. The constant MMDB_STORAGE_TYPE_EXTERNAL (value=0) indicates that the updated object will be stored in a server file.

overwrite (INTEGER)
A value that indicates whether to overwrite the target file if it already exists. The value can be 0 or 1. A value of 0 means the target file will not be overwritten (in effect, the retrieval will not take place). A value of 1 means the target file will be overwritten if the target file already exists.

target_format (VARCHAR(8))
The format of the image after retrieval or update. The format of the source image will be converted as appropriate. For retrieval of an

image to a server file, if the target_file is the same as the source_file, the target format must be the same as the source format. For MPG1 format, you can specify MPG1, mpg1, MPEG1, or mpeg1. For MPG2 format, you can specify MPG2, mpg2, MPEG2, or mpeg2.

conversion_options (VARCHAR(100))
Specifies changes, such as rotation and compression, to be applied to the image when it is retrieved or updated. See Table 6 on page 83 for the supported conversion options.

content (BLOB(2G) AS LOCATOR)
The host variable that contains the content for update of the image, audio, or video. The host variable can be of type BLOB, BLOB_FILE, or BLOB_LOCATOR. DB2 promotes the data type of the content to BLOB_LOCATOR and passes the LOB locator to the Content UDF.

source_format (VARCHAR(8))
The format of the source for update of the image, audio, or video. A null value or empty string can be specified, or for image only, the character string ASIS; in these three cases, the extender will attempt to determine the format automatically. For MPG1 format, you can specify MPG1, mpg1, MPEG1, or mpeg1. For MPG2 format, you can specify MPG2, mpg2, MPEG2, or mpeg2.

attrs (LONG VARCHAR FOR BIT)
The attributes of the image, audio, or video

thumbnail (LONG VARCHAR FOR BIT DATA)
A thumbnail of the image or video frame (image and video only)

Return values (data type)

The content of the retrieved image, audio, or video if retrieved to a buffer, (BLOB(2G) AS LOCATOR). If retrieved to a file, VARCHAR(254).

For update, the handle of the image, audio, or video to be updated (DB2IMAGE, DB2AUDIO, or DB2VIDEO).

Examples

Retrieve into a server file the image stored for Anita Jones in the picture column of the employee table:

```
struct{
      short len;
      char data[250];
      }hvImg_fname;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT CONTENT (PICTURE,
```

```
       '/employee/images/ajones.bmp',1)
      INTO :hvImg_fname
      FROM EMPLOYEE
      WHERE NAME='Anita Jones';
```

Retrieve into a client buffer the 1-MB audio clip stored for Robert Smith in the sound column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE IS BLOB_LOCATOR audio_loc;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT CONTENT (SOUND, 1, 1000000)
      INTO :audio_loc
      FROM EMPLOYEE
      WHERE NAME='Robert Smith';
```

Update Anita Jones's image in the picture column of the employee table; convert the format of the image from BMP to GIF and reduce the image to 50% of its original size:

```
EXEC SQL UPDATE EMPLOYEE
  SET picture = CONTENT(PICTURE,
       '/employee/newimg/ajones.bmp',
       'BMP',
       'GIF',
        '-s 0.5',
       '');
  WHERE NAME='Anita Jones';
```

DB2Audio

Image	Audio	Video
	X	

Stores the content of an audio in a database table. The audio source can be in a client buffer, client file, or server file. The audio can be stored in the database table as a BLOB, or in a server file (referenced by the database table). The audio source can be in a supported format, in which case, the DB2Audio Extender identifies its attributes for storage, or in an unsupported format, in which case the attributes must be specified in the UDF.

Include file

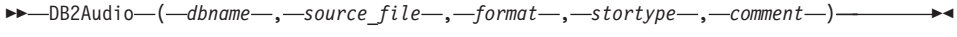
dmbaudio.h

Syntax

Store content from buffer or client file

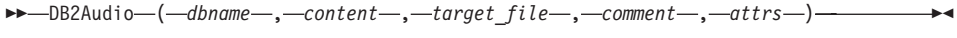
Syntax

Store content from server file



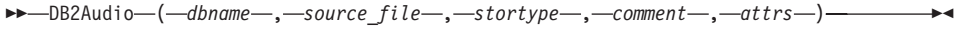
Syntax

Store content with user-supplied attributes from buffer or client file



Syntax

Store content with user-supplied attributes from server file



Parameters (data type)

dbname (VARCHAR(18))
The name of the currently connected database, as indicated by the CURRENT SERVER special register.

content (BLOB(2G) AS LOCATOR)
The host variable that contains the content of the audio. The host variable can be of type BLOB, BLOB_FILE, or BLOB_LOCATOR. DB2 promotes the data type of the content to BLOB-LOCATOR and passes the LOB locator to the DB2Audio UDF.

format (VARCHAR(8))
The format of the source audio. A null value or empty string can be specified, in which case the Audio Extender will attempt to determine the source format automatically. The audio will be stored in the same format as its source. See Table 5 on page 81 for supported audio formats.

target_file (LONG VARCHAR)
The name of the target server file (for storage to a server file), or a null value or empty string (for storage into a database table as a BLOB). The target file can be a fully qualified name. If the name is unqualified, the DB2AUDIOSTORE and DB2MMSTORE environment variables on the server are used to locate the file.

source_file (LONG VARCHAR)
The name of the source server file. The source file name can be a fully qualified name or an unqualified name; it cannot be a null value or empty string. If the name is unqualified, the DB2AUDIOPATH and DB2MMPATH environment variables on the server will be used to locate the file.

stortype (INTEGER)
A value that indicates where the audio will be stored. The constant MMDB_STORAGE_TYPE_INTERNAL (value=1) indicates that the audio will be stored in the database as a BLOB; the constant MMDB_STORAGE_TYPE_EXTERNAL (value=0) indicates that the audio content will be stored in a server file (pointed to from the database).

comment (LONG VARCHAR)

A comment to be stored with the audio.

attrs (LONG VARCHAR FOR BIT DATA)
The attributes of the audio.

Return values (data type)

Handle of the audio (DB2AUDIO)

Examples

Insert a record that includes an audio clip for Anita Jones into the employee table. The audio source is in a client buffer. Store the audio clip in the table as a BLOB:

```
EXEC SQL BEGIN DECLARE SECTION;
     SQL TYPE IS BLOB (5M) aud_seg;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
      '128557',
      'Anita Jones',
      DB2AUDIO(
         CURRENT SERVER,
         :aud_seg,
         'WAVE',
         CAST(NULL as LONG VARCHAR),

         'Anita''s voice'));
```

Insert a record that includes an audio clip for Robert Smith into the employee table. The audio source is in a server file. The employee table record will point to the file.

```
EXEC SQL BEGIN DECLARE SECTION;
     long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType = MMDB_STORAGE_TYPE_EXTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
      '384779',
      'Robert Smith',
      DB2AUDIO(
         CURRENT SERVER,
         '/employee/sounds/rsmith.wav',
         'WAV',
         :hvStorageType,
         'Robert''s voice'));
```

Insert a record that includes an audio clip for Anita Jones into the employee table. Store the audio clip as a BLOB. The source audio clip, which is in a server file, has a user-defined format, a sampling rate of 44.1 KHz, and has two recorded channels.

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
struct {
       short len;
       char data[600];
}hvAudattr;
EXEC SQL END DECLARE SECTION;
```

```
MMDBAudioAttrs          *paudiattr;

hvStorageType = MMDB_STORAGE_TYPE_INTERNAL;

paudioattr=(MMDBAudioAttrs *) hvAudattr.data;
strcpy(paudioAttr->cFormat,"cFormatA");
paudioAttr->ulSamplingRate=44100;
paudioAttr->usNumChannels=2;
hvAudattrs.len=sizeof(MMDBAudioAttrs);

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
               DB2AUDIO(
               CURRENT SERVER,
               '/employee/sounds/ajones.aud',
               :hvStorageType,
               'Anita"s voice',
               :hvAudattr)
       );
```

DB2Image

Image	Audio	Video
X		

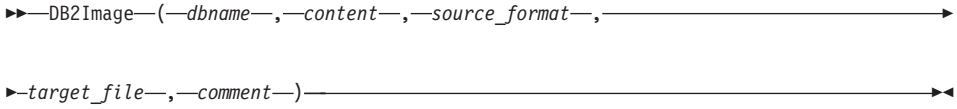
Stores the content of an image in a database table. The image source can be in a client buffer, client file, or server file. The image can be stored in the database table as a BLOB, or in a server file (referenced by the database table). The image source can be in a supported format, in which case the DB2 Image Extender identifies its attributes for storage, or in an unsupported format, in which case the attributes must be specified in the UDF.

Include file

dmbimage.h

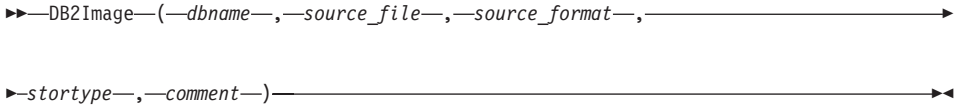
Syntax

Store content from buffer or client file



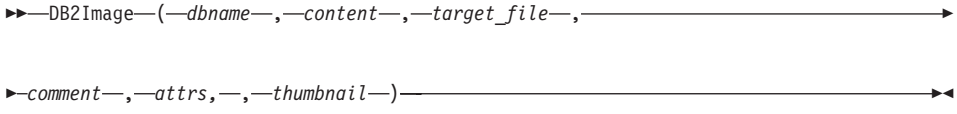
Syntax

Store content from server file



Syntax

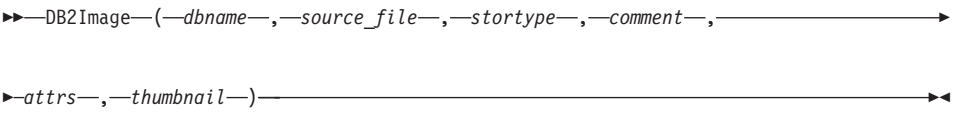
Store content with user-supplied attributes from buffer or client file



DB2Image

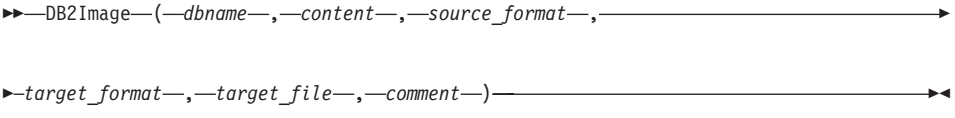
Syntax

Store content with user-supplied attributes from server file



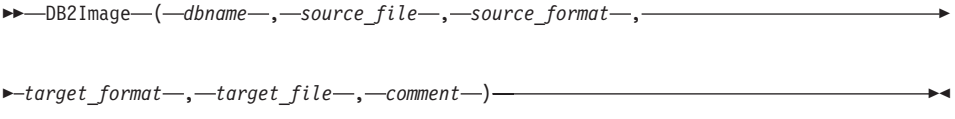
Syntax

Store content from buffer or client file with format conversion



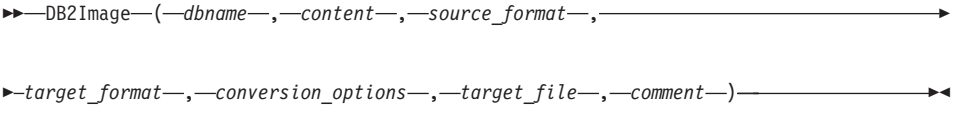
Syntax

Store content from server file with format conversion



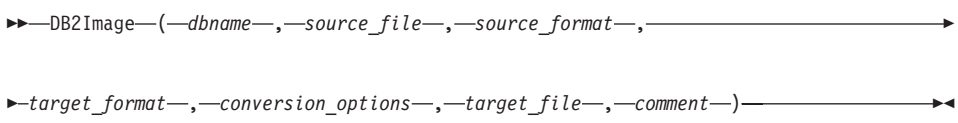
Syntax

Store content from buffer or client file with format conversion and additional changes



Syntax

Store content from server file with format conversion and additional changes



Parameters (data type)

dbname (VARCHAR(18))
The name of the currently connected database, as indicated by the CURRENT SERVER special register.

content (BLOB(2G) AS LOCATOR)
The host variable that contains the content of the image. The host variable can be of type BLOB, BLOB_FILE, or BLOB_LOCATOR. DB2 promotes the data type of the content to BLOB_LOCATOR and passes the LOB locator to the DB2Image UDF.

source_format (VARCHAR(8))
The format of the source image. A null value, empty string, or the character string ASIS can be specified; in any of these three cases, the Image Extender will attempt to determine the source format automatically. The image will be stored in the same format as its source. See Table 5 on page 81 for supported image formats.

target_format (VARCHAR(8))
The format of the image after storage. The format of the source image will be converted as appropriate.

target_file (LONG VARCHAR)
The name of the target server file (for storage to a server file), or a null value or empty string (for storage into a database table as a BLOB). The target file name can be a fully qualified name. If the name is unqualified, the DB2IMAGESTORE and DB2MMSTORE environment variables on the server are used to locate the file. If the image is stored with format conversion, the path to the target file needs to be specified in the DB2IMAGEPATH and DB2MMPATH environment variables.

source_file (LONG VARCHAR)
The name of the source server file. The source file name can be a fully qualified name or an unqualified name; it cannot be a null value or empty string. If the name is unqualified, the DB2IMAGEPATH and DB2MMPATH environment variables on the server will be used to locate the file.

stortype (INTEGER)
A value that indicates where the image will be stored. The constant MMDB_STORAGE_TYPE_INTERNAL (value=1) indicates that the

image will be stored in the database as a BLOB; the constant MMDB_STORAGE_TYPE_EXTERNAL (value=0) indicates that the image content will be stored in a server file (pointed to from the database).

comment (LONG VARCHAR)
A comment to be stored with the image.

attrs (LONG VARCHAR FOR BIT DATA)
The attributes of the image.

thumbnail (LONG VARCHAR FOR BIT DATA)
A thumbnail of the image.

conversion_options (VARCHAR(100))
Specifies changes, such as rotation and compression, to be applied to the image when it is stored. See Table 6 on page 83 for the supported conversion options.

Return values (data type)

Handle of the image (DB2IMAGE)

Examples

Insert a record that includes an image for Anita Jones into the employee table. The image source is in a client buffer. Store the image in the table as a BLOB:

```
EXEC SQL BEGIN DECLARE SECTION
     SQL TYPE IS BLOB (2M) hvImg
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
      '128557',
      'Anita Jones',
      DB2IMAGE(
         CURRENT SERVER,
         :hvImg,
         'ASIS',
         CAST(NULL as LONG VARCHAR),

         'Anita''s picture'));
```

Insert a record that includes an image for Robert Smith into the employee table. The image source is in a server file. The employee table record will point to the file. Convert the format of the image from BMP to GIF when stored. Also crop the image to a width of 110 pixels and a height of 150 pixels and compress the image using LZW type compression:

```
EXEC SQL INSERT INTO EMPLOYEE VALUES(
      '384779',
      'Robert Smith',
```

```
      DB2IMAGE(
         CURRENT SERVER,
         '/employee/pictures/rsmith.bmp',
         'BMP',
         'GIF',
         '-x 110 -y 150 -c 14',
         '',
         'Robert"s picture'));
```

Insert a record that includes an image for Robert Smith into the employee table. The source image, which is in a server file, has a user-defined format, a height of 640 pixels, and a width of 480 pixels. Store the image as a BLOB:

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
struct {
       short len;
       char data[400];
}hvImgattrs;
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType = MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs.data;
strcpy(pimgattrt→cFormat,"FormatI");
pimgattr→width=640;
pimgattr→height=480;
hvImgattrs.len=sizeof(DB2IMAGEATTRS);

EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2IMAGE(
          CURRENT SERVER,
          '/employee/images/ajones.bmp',
          :hvStorageType,
          'Anita''s picture',
          :hvImgattrs,
          CAST(NULL as LONG VARCHAR))
       );
```

DB2Video

Image	Audio	Video
		X

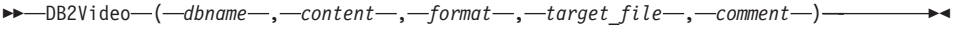
Stores the content of a video in a database table. The video source can be in a client buffer, client file, or server file. The video can be stored in the database table as a BLOB, or in a server file (referenced by the database table). The video source can be in a supported format, in which case the DB2 Video Extender identifies its attributes for storage, or in an unsupported format, in which case the attributes must be specified in the UDF.

Include file

dmbvideo.h

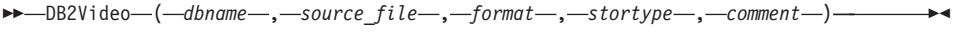
Syntax

Store content from buffer or client file



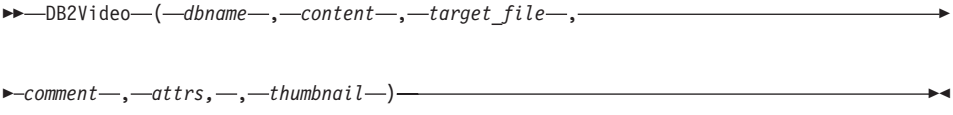
Syntax

Store content from server file



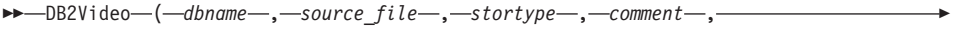
Syntax

Store content with user-supplied attributes from buffer or client file



Syntax

Store content with user-supplied attributes from server file



Parameters (data type)

dbname (VARCHAR(18))
The name of the currently connected database, as indicated by the CURRENT SERVER special register.

content (BLOB(2G) AS LOCATOR)
The host variable that contains the content of the video. The host variable can be of data type BLOB, BLOB_FILE, or BLOB_LOCATOR. DB2 promotes the content to BLOB_LOCATOR and passes the LOB locator to the DB2Video UDF. If the content is in a client buffer, the buffer must contain at least the first 640 KB of the content to ensure that the complete video header is read.

format (VARCHAR(8))
The format of the source video. A null value or empty string can be specified, in which case, the Video Extender will attempt to determine the source format automatically. The video will be stored in the same format as its source.See Table 5 on page 81 for supported video formats. For MPG1 format, you can specify MPG1, mpg1, MPEG1, or mpeg1. For MPG2 format, you can specify MPG2, mpg2, MPEG2, or mpeg2.

target_file (LONG VARCHAR)
The name of the target server file (for storage to a server file), or a null value or empty string (for storage into a database table as a BLOB). The server file can must be a fully qualified name. If the file name is unqualified, the DB2VIDEOSTORE and DB2MMSTORE environment variables on the server are used to locate the file.

source_file (LONG VARCHAR)
The name of the source server file. The name can be a fully qualified name or an unqualified name; it cannot be a null value or empty string. If the name is unqualified, the DB2VIDEOPATH and DB2MMPATH environment variables on the server will be used to locate the file.

stortype (INTEGER)
A value that indicates where the video will be stored. The constant MMDB_STORAGE_TYPE_INTERNAL (value=1) indicates that the video will be stored in the database as a BLOB; the constant MMDB_STORAGE_TYPE_EXTERNAL (value=0) indicates that the video content will be stored in a server file (pointed to from the database).

comment (LONG VARCHAR)
A comment to be stored with the video.

attrs (LONG VARCHAR FOR BIT DATA)
The attributes of the video.

thumbnail (LONG VARCHAR FOR BIT DATA)
A thumbnail image that represents the video.

Return values (data type)

Handle of the video (DB2VIDEO)

Examples

Insert a record that includes a video clip for Anita Jones into the employee table. The video source is in a client buffer. Store the video clip in the table as a BLOB:

```
EXEC SQL BEGIN DECLARE SECTION
     SQL TYPE IS BLOB (8M) vid;
EXEC SQL END DECLARE SECTION;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
      '128557',
      'Anita Jones',
      DB2VIDEO(
         CURRENT SERVER,
         :vid,
         'MPEG1',
         CAST(NULL as LONG VARCHAR),
         'Anita''s video'));
```

Insert a record that includes a video clip for Robert Smith into the employee table. The video source is in a server file. The employee table record will point to the file:

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType = MMDB_STORAGE_TYPE_EXTERNAL;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
      '384779',
      'Robert Smith',
      DB2VIDEO(
         CURRENT SERVER,
         '/employee/videos/rsmith.mpg',
         'MPEG1',
         :hvStorageType,
         'Robert''s video'));
```

Insert a record that includes a video clip in a database table. The source video clip, which is in a server file, has a user-defined format. Keep the video content in the server file (the database table record will point to the file). Also store a thumbnail that represents the video:

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
struct {
       short len;
       char data[400];
}hvVidattrs;
struct {
       short len;
       char data[10000];
}hvThumbnail;
EXEC SQL END DECLARE SECTION;

MMDBVideoAttrs         *pvideoAttr;

hvStorageType = MMDB_STORAGE_TYPE_EXTERNAL;

pvideoAttr=(MMDBVideoAttrs *)hvVidattrs.data;
strcpy(pvideoAttr->cFormat,"Formatv");
pvideoAttr.len=sizeof(MMDBVideoAttrs);
:
/* Generate thumbnail and assign data   */
/* in video structure                   */
:
EXEC SQL INSERT INTO EMPLOYEE VALUES(
        '128557',
        'Anita Jones',
        DB2VIDEO(
          CURRENT SERVER,
          '/employee/videos/ajones.vid',
          :hvStorageType,
          'Anita''s video',
          :hvVidattrs,
          :hvThumbnail)
       );
```

Duration

Image	Audio	Video
	X	X

Returns the duration (that is, playing time in seconds) of a WAVE or AIFF audio, or video.

Include file

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2AUDIO or DB2VIDEO)
Column name or host variable that contains the handle of the audio or video.

Return values (data type)

Duration, in seconds, of a video, or a WAVE, AIFF or user-defined format audio (INTEGER). Returns a null value for audio in other formats.

Examples

Display the duration of all videos stored in the video column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
long hvDur_vid;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT DURATION(VIDEO)
     INTO :hvDur_vid
     FROM EMPLOYEE;
```

Filename

Image	Audio	Video
X	X	X

Returns the name of the server file that contains the contents of an image, audio, or video if the object content is stored in a file (pointed to from a database table). If the image, audio, or video is stored in a database table as a BLOB, a null value is returned.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

Return values (data type)

File name of the server file if object content is in a server file (VARCHAR(250)); null value if object is stored as a BLOB.

Examples

Display the file name of the video for the Robert Smith entry in the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvVid_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(VIDEO)
     INTO :hvVid_fname
     FROM EMPLOYEE
     WHERE NAME='Robert Smith';
```

FindInstrument

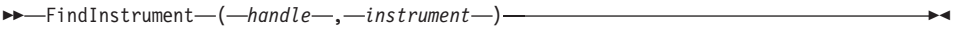
Image	Audio	Video
	X	

Returns the track number of the first occurrence of a specified instrument in a MIDI audio.

Include file

dmbaudio.h

Syntax



Parameters (data type)

handle (DB2AUDIO)
Column name or host variable that contains the handle of the audio.

instrument (VARCHAR(255))
Name of the instrument to be searched for. The Audio Extender will look for an instrument whose name exactly matches the supplied name.

Return values (data type)

Track number that contains the first occurrence of the specified instrument name (SMALLINT); a value of -1 is returned if an instrument of the specified name is not found. NULL is returned for audio in other formats.

Examples

Find the first occurrence of PIANO in Robert Smith's MIDI audio recording stored in the sound column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
      short hvInstr;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FINDINSTRUMENT(SOUND, 'PIANO')
      INTO :hvInstr
      FROM EMPLOYEE
      WHERE NAME = 'Robert Smith';
```

FindTrackName

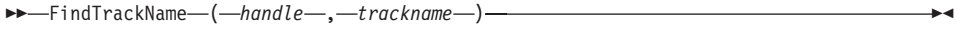
Image	Audio	Video
	X	

Returns the number of a specified named track in a MIDI audio.

Include file

dmbaudio.h

Syntax



Parameters (data type)

handle (DB2AUDIO)
Column name or host variable that contains the handle of the audio.

trackname (VARCHAR(255))
Name of the track to be searched for. The Audio Extender will look for a track whose name exactly matches the supplied name.

Return values (data type)

Number of the named track; of the specified instrument name (SMALLINT). A value of -1 is returned if a track of the specified name is not found. A null value is returned for audio in other formats.

Examples

Determine if there is a track named WELCOME in Robert Smith's MIDI audio recording stored in the sound column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
      short hvTrack;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FINDTRACKNAME(SOUND,
          'WELCOME')
      INTO :hvTrack
      FROM EMPLOYEE
      WHERE NAME = 'Robert Smith';
```

Format

Image	Audio	Video
X	X	X

Returns the format of an image, audio, or video.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

Return values (data type)

Format of the image, audio, or video (VARCHAR(8). See Table 5 on page 81 for the supported image, audio, and video formats.

Examples

Get the names of all employees whose images stored in the picture column of the employee table are in GIF format:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvName[30];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT NAME
     INTO :hvName
     FROM EMPLOYEE
     WHERE FORMAT(PICTURE)='GIF';
```

FrameRate

Image	Audio	Video
		X

Returns the throughput of a video in frames per second.

Include file

dmbvideo.h

Syntax

Parameters (data type)

handle (DB2VIDEO)
Column name or host variable that contains the handle of the video.

Return values (data type)

Frame rate of video (SMALLINT). Returns a null value if the throughput rate is variable.

Examples

Get the frame rate of the video stored in the video column of the employee table for Anita Jones:

```
EXEC SQL BEGIN DECLARE SECTION;
short hvFm_rate;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FRAMERATE (VIDEO)
     FROM EMPLOYEE
      INTO :hvFm_rate
     WHERE NAME='Anita Jones';
```

GetInstruments

Image	Audio	Video
	X	

Returns instrument name of all instruments in a MIDI audio.

Include file

dmbaudio.h

Syntax

Parameters (data type)

handle (DB2AUDIO)
Column name or host variable that contains the handle of the audio.

Return values (data type)

Instrument name of all instruments in the MIDI audio (VARCHAR(1536)). The values are returned in track number order (for example, PIANO; TRUMPET; BASS). The result is divided into *n* fields, where *n* is the number of tracks in the MIDI audio. If a track does not have an associated intrument, its field is blank. A null value is returned for audio formats other than MIDI.

Examples

Find all the instruments (that is, track numbers and instrument names) in Robert Smith's MIDI audio recording stored in the sound column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_Instr[1536];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT GETINSTRUMENTS(SOUND)
      INTO :hvAud_Instr
      FROM EMPLOYEE
      WHERE NAME = 'Robert Smith';
```

GetTrackNames

Image	Audio	Video
	X	

Returns the name of all tracks in a MIDI audio.

Include file

dmbaudio.h

Syntax

Parameters (data type)

handle (DB2AUDIO)
Column name or host variable that contains the handle of the audio.

Return values (data type)

Name of all tracks in the MIDI audio (VARCHAR(1536)). The values are returned in track number order (for example, PIANO TUNE; TRUMPET FANFARE). The result is divided into *n* fields, where *n* is the number of tracks in the MIDI audio. If a track does not have a name, its field is blank. A null value is returned for audio formats other than MIDI.

Examples

Get all the track numbers and track names in Robert Smith's MIDI audio recording stored in the sound column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvTracks[1536];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT GETTRACKNAMES(SOUND)
      INTO :hvTracks
      FROM EMPLOYEE
      WHERE NAME = 'Robert Smith';
```

Height

Image	Audio	Video
X		X

Returns the height, in pixels, of an image or video frame.

Include file

image dmbimage.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE or DB2VIDEO)
Column name or host variable that contains the handle of the image or video.

Return values (data type)

Height in pixels (INTEGER)

Examples

Get the file name of all images in the picture column of the employee table that are shorter than 500 pixels:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvImg_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE HEIGHT(PICTURE)<500;
```

Importer

Image	Audio	Video
X	X	X

Returns the user ID of the person who stored an image, audio, or video in a database table.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

Return values (data type)

User ID of importer (CHAR(8))

Examples

Get the name of all files for audios stored in the sound column of the employee table by user ID rsmith:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
     INTO :hvAud_fname
     FROM EMPLOYEE
     WHERE IMPORTER(SOUND)='rsmith';
```

ImportTime

Image	Audio	Video
X	X	X

Returns a timestamp that indicates when an image, audio, or video was stored in a database table.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

Return values (data type)

Timestamp when image, audio, or video was stored (TIMESTAMP)

Examples

Get the names of all files for images that were stored in the picture column of the employee table more than a year ago:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvImg_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE(CURRENT TIMESTAMP -
        IMPORTTIME(PICTURE))>365;
```

MaxBytesPerSec

Image	Audio	Video
		X

Returns the maximum throughput of a video in bytes per second.

Include file

dmbvideo.h

Syntax

Parameters (data type)

handle (DB2VIDEO)
Column name or host variable that contains the handle of the video.

Return values (data type)

Throughput of video (INTEGER). Returns a null value if the throughput rate is variable.

Examples

Get the maximum throughput of the video stored in the video column of the employee table for Anita Jones:

```
EXEC SQL BEGIN DECLARE SECTION;
long hvMax_BytesPS;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT MAXBYTESPERSEC(VIDEO)
     INTO :hvMax_BytesPS
     FROM EMPLOYEE
     WHERE NAME='Anita Jones';
```

NumAudioTracks

Image	Audio	Video
	X	X

Returns the number of audio tracks in a video or MIDI audio.

Include file

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2AUDIO or DB2VIDEO)
Column name or host variable that contains the handle of the audio or video.

Return values (data type)

Number of audio tracks in the video or MIDI audio (SMALLINT). Returns a null value for audio in other formats.

Examples

Get the names of any video files from the video column of the employee table that do not contain any audio tracks:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvVid_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(VIDEO)
     INTO :hvVid_fname
     FROM EMPLOYEE
     WHERE NUMAUDIOTRACKS(VIDEO) = 0;
```

NumChannels

Image	Audio	Video
	X	X

Returns the number of recorded audio channels in a WAVE or AIFF audio, or video.

Include file

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2AUDIO or DB2VIDEO)
Column name or host variable that contains the handle of the audio or video.

Return values (data type)

Number of recorded audio channels in the video or WAVE or AIFF audio (SMALLINT). Returns a null value for audio in other formats.

Examples

Get the names of all audio files from the sound column of the employee table that were recorded in stereo (that is, 2 channels):

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
     INTO :hvAud_fname
     FROM EMPLOYEE
     WHERE NUMCHANNELS(SOUND) = 2;
```

NumColors

Image	Audio	Video
X		

Returns the number of colors in an image.

Include file

dmbimage.h

Syntax

Parameters (data type)

handle (DB2IMAGE)
Column name or host variable that contains the handle of the image.

Return values (data type)

Number of colors in image (INTEGER)

Examples

Get the names of image files from the picture column of the employee table for images that have less than 16 colors:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvImg_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE NUMCOLORS(PICTURE) < 16;
```

NumFrames

Image	Audio	Video
		X

Returns the number of frames in a video.

Include file

dmbvideo.h

Syntax

Parameters (data type)

handle (DB2VIDEO)
Column name or host variable that contains the handle of the video.

Return values (data type)

Number of frames in video (INTEGER). Returns a null value if the throughput rate is variable.

Examples

Get the number of frames in the video stored in the video column of the employee table for Robert Smith:

```
EXEC SQL BEGIN DECLARE SECTION;
long hvNum_Frames;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT NUMFRAMES (VIDEO)
     INTO :hvNum_Frames
     FROM EMPLOYEE
     WHERE NAME='Robert Smith';
```

NumVideoTracks

Image	Audio	Video
		X

Returns the number of video tracks in a video.

Include file

dmbvideo.h

Syntax

Parameters (data type)

handle (DB2VIDEO)
Column name or host variable that contains the handle of the video.

Return values (data type)

Number of video tracks (SMALLINT)

Examples

Get the file name of all videos from the video column of the employee table that have more than one video track:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvVid_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME (VIDEO)
     INTO :hvVid_fname
     FROM EMPLOYEE
     WHERE NUMVIDEOTRACKS(VIDEO) > 1;
```

QbScoreFromName

Image	Audio	Video
X		

Returns the score of an image, which is a number that expresses how closely the features of the image match those of a query object. The QBIC catalog associated with the column to which the image handle belongs is used to calculate the score of the image. The lower the score, the more closely the features of the image match those of the specified query object. (QbScoreFromName replaces QbScore, but QbScore is still accepted.)

Include file

none

Syntax

Parameters (data type)

queryName (varchar(18))
The name of the query object.

imgHandle (DB2Image)
The handle of the image.

Return values (data type)

The score of the image (DOUBLE). The score can range from 0.0 to a very large number approaching infinity. The lower the score, the closer the feature values of the target image match the feature values specified in the query. A score of 0.0 means an exact match. A score of -1 means that the image has not been cataloged.

Examples

Find the cataloged images in a table column whose average color is very close to red:

```
EXEC SQL BEGIN DECLARE SECTION;
char Img_hdl[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT NAME
  INTO :Img_hdl
```

```
FROM FABRIC
WHERE (QBSCOREFROMNAME(
         'fshavgcol',
          SWATCH_IMG))<0.1;
```

QbScoreFromStr

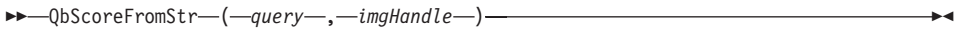
Image	Audio	Video
X		

Returns the score of an image, which is a number that expresses how closely the features of the image match those of a query string. The QBIC catalog that is associated with the column to which the image handle belongs is used to calculate the score of the image. The lower the score, the more closely the features of the image match those of the query string.

Include file

none

Syntax



Parameters (data type)

query (VARCHAR(1024))
The query string.

imgHandle (DB2Image)
The handle of the image.

Return values (data type)

The score of the image (DOUBLE). The score can range from 0.0 to a very large number approaching infinity. The lower the score, the more closely the feature values of the target image match the feature values specified in the query. A score of 0.0 means an exact match. A score of -1 means that the image has not been cataloged.

Examples

Find the cataloged images in a table column whose average color is very close to red.:

```
SELECT name
FROM fabric
WHERE (QbScoreFromStr(
            'QbColorFeatureClass color=<255, 0, 0>',
             Swatch_Img))<0.1
```

QbScoreTBFromName

Image	Audio	Video
X		

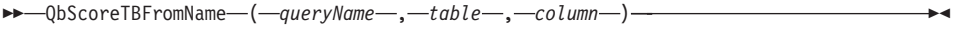
Returns a table of scores for an image column. Each score is a number that expresses how closely the features of the image match those of the query object. The QBIC catalog that is associated with the specified table and column to which the image handle belongs is used to calculate the score of each image. The lower the score for any image, the more closely the features of that image match those of the query object.

Include file

none

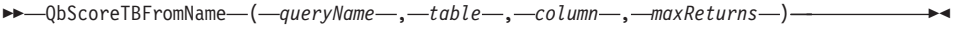
Syntax

Return scores for all cataloged images in a column



Syntax

Return scores for a specific number of cataloged images in a column



Parameters (data type)

queryName (VARCHAR(18))
The name of the query object.

table (CHAR(18))
The name of the table that contains the image column.

column (CHAR(18))
The name of the image column.

maxReturns (INTEGER)
The maximum number of handles that the table of results is to return. If a value is not specified, the maximum number of handles returned is 100.

Return values (data type)

Table of image handles and scores for the images in the column. The result table has two columns: IMAGE_ID (DB2Image) which contains the image handles, and SCORE (DOUBLE) which contains the scores. The result table is arranged in descending order by score. The score can range from 0.0 to a very large number approaching infinity. The lower the score, the closer the feature values of the target image match the feature values specified in the query. A score of 0.0 means an exact match. A score of -1 means that the image has not been cataloged.

Examples

Compare the texture of the images in a table column to the texture specified in a query object; return the image handles and their scores:

```
SELECT name, description
INTO :hvName, :hvDesc
FROM fabric
WHERE CAST (swatch_img as varchar(250)) IN
 (SELECT CAST (image_id as varchar(250)) FROM TABLE
 (QbScoreTBFromName
     'fstxtr',
     'fabric',
     'swatch_img'))
 AS T1));
```

QbScoreTBFromStr

Image	Audio	Video
X		

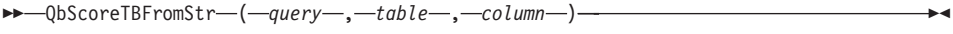
Returns a table of scores from an image column. Each score is a number that expresses how closely the features of the image are to those specified in a query string. The QBIC catalog that is associated with the table and column to which the image handle belongs is used to calculate the score of each image. The lower the score for an image, the more closely the features of that image match those of the query string.

Include file

none

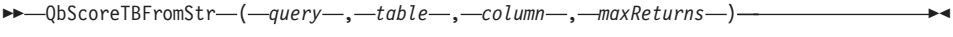
Syntax

Return scores for all cataloged images in a column



Syntax

Return scores for a specific number of cataloged images in a column



Parameters (data type)

query (VARCHAR(1024))
The query string.

table (CHAR (18))
The table containing the image column.

column (CHAR(18))
The image column to query.

maxReturns (INTEGER)
The maximum number of handles that the table of results is to return. If a value is not specified, the maximum number of image handles returned is 100.

Return values (data type)

Table of image handles and scores for the images in the column. The result table has two columns: IMAGE_ID (DB2Image) which contains the image handles, and SCORE (DOUBLE) which contains the scores. The result table is arranged in descending order by score. The score can range from 0.0 to a very large number approaching infinity. The lower the score, the closer the feature values of the target image match the feature values specified in the query. A score of 0.0 means an exact match. A score of -1 means that the image has not been cataloged.

Examples

Find the ten cataloged images in a table column whose texture is closest to that of an image in a server file:

```
SELECT name, description
FROM fabric
WHERE CAST (swatch_img as varchar(250)) IN
 (SELECT CAST (image_id as varchar(250)) FROM TABLE
 (QbScoreTBFromStr
 (QbTextureFeatureClass file=<server,"patterns/ptrn07.gif">'
      'fabric',
      'swatch_img',
      10))
  AS T1));
```

Replace

Image	Audio	Video
X	X	X

Updates the content of an image, audio, or video stored in a database, and updates its comment.

Include file

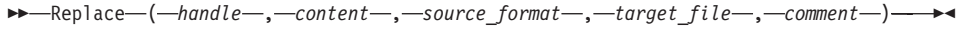
image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

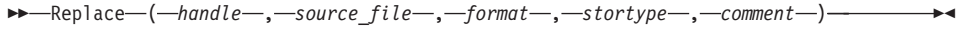
Syntax

Update content from buffer or client file and update comment



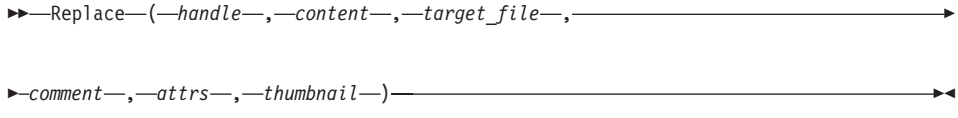
Syntax

Update content from server file and update comment



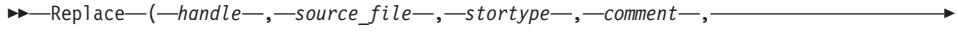
Include file

Update content with user-supplied attributes from buffer or client file and update comment



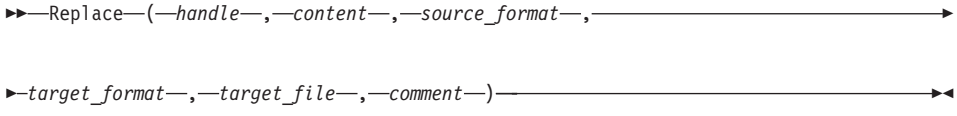
Include file

Update content with user-supplied attributes from server file and update comment



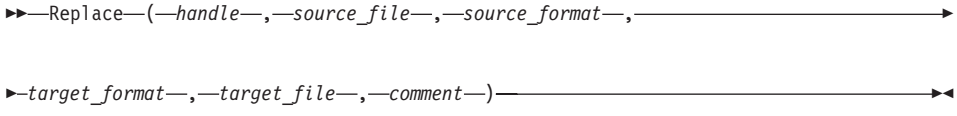
Syntax

Update content from buffer or client file with format conversion and update comment—image only



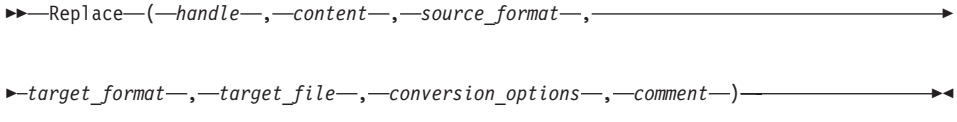
Syntax

Update content from server file with format conversion and update comment—image only



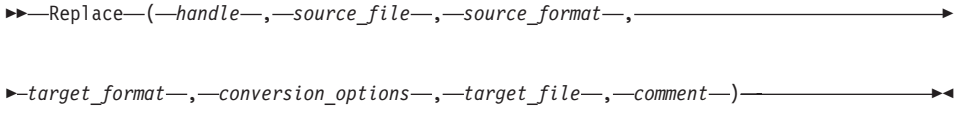
Syntax

Update content from buffer or client file with format conversion and additional changes and update comment—image only



Syntax

Update content from server file with format conversion and additional changes and update comment—image only



Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

source_file (LONG VARCHAR)
The name of the file that contains the content for the update of the image, audio, or video.

target_file (LONG VARCHAR)
The name of the file that contains the content of the image, audio, or video to be updated.

create_target (INTEGER)
A value that indicates whether a target file is to be created if the source content is in a server file. The value can be 0 or 1. A value of 0 means the target file will not be created (in effect, the retrieval will not take place). A value of 1 means the target file will be created (if the target file already exists, the effect of this value is to overwrite the file). If the source content is a BLOB, the target file is always created (if the file already exists, it is overwritten).

target_format (VARCHAR(8))
The format of the image after retrieval. The format of the source image will be converted as appropriate. If the content is updated with format conversion, the path to the target file needs to be specified in the DB2IMAGEPATH and DB2MMPATH environment variables. For MPG1 format, you can specify MPG1, mpg1, MPEG1, or mpeg1. For MPG2 format, you can specify MPG2, mpg2, MPEG2, or mpeg2.

content (BLOB(2G) AS LOCATOR)
The host variable that contains the content for update of the image, audio, or video. The host variable can be of type BLOB, BLOB_FILE, or BLOB_LOCATOR. DB2 promotes the data type to BLOB_LOCATOR and passes the LOB locator to the Replace UDF.

source_format (VARCHAR(8))
The format of the source for update of the image, audio, or video. A null value or empty string can be specified, or for image only, the character string ASIS; in these three cases, the extender attempts to determine the format automatically. For MPG1 format, you can specify MPG1, mpg1, MPEG1, or mpeg1. For MPG2 format, you can specify MPG2, mpg2, MPEG2, or mpeg2.

comment (LONG VARCHAR)
A comment.

attrs (LONG VARCHAR FOR BIT DATA)
The attributes of the image, audio, or video

thumbnail (LONG VARCHAR FOR BIT DATA)
A thumbnail of the image or video frame (image and video only)

conversion_options (VARCHAR(100))
Specifies changes, such as rotation and compression, to be applied to the image when it is updated. See Table 6 on page 83 for the supported conversion options.

Return values (data type)

The handle of the image, audio, or video to be updated (DB2IMAGE, DB2AUDIO, or DB2VIDEO).

Examples

Update Anita Jones's image in the picture column of the employee table, convert the format of the image from BMP to GIF, and update the comment:

```
EXEC SQL BEGIN DECLARE SECTION;
  long hvStorageType;
EXEC SQL END DECLARE SECTION;

hvStorageType = MMDB_STORAGE_TYPE_INTERNAL;

EXEC SQL UPDATE EMPLOYEE
  SET PICTURE = REPLACE(PICTURE,
        '/employee/newimg/ajones.bmp',
        'BMP',
        'GIF',
        :hvStorageType,
        'Anita''s new picture')
  WHERE NAME='Anita Jones';
```

SamplingRate

Image	Audio	Video
	X	X

Returns the sampling rate of a WAVE or AIFF audio, or of an audio track in a video, in number of samples per second.

Include file

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2AUDIO or DB2VIDEO)
Column name or host variable that contains the handle of the audio or video.

Return values (data type)

Sampling rate of video or WAVE or AIFF audio (INTEGER). Returns a null value for audio in other formats.

Examples

Get the file name of all audios from the sound column of the employee table whose sampling rate is 44.1 KHz:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME (SOUND)
     INTO :hvAud_fname
     FROM EMPLOYEE
     WHERE SAMPLINGRATE(SOUND) = 44100;
```

Size

Image	Audio	Video
X	X	X

Returns the size of an image, audio, or video, in bytes.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

Return values (data type)

Size, in bytes, of image, audio, or video (INTEGER).

Examples

Get the file name of all images in the picture column of the employee table whose size is greater than 310 KB:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvImg_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE SIZE(PICTURE) > 310000;
```

Thumbnail

Image	Audio	Video
X		X

Returns or updates a thumbnail-size version of an image or video frame stored in a database.

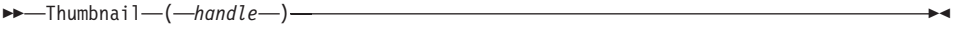
Include file

image dmbimage.h

video dmbvideo.h

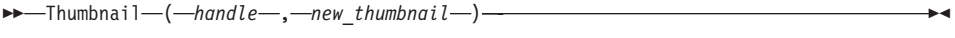
Syntax

Retrieve a thumbnail



Syntax

Update a thumbnail



Parameters (data type)

handle (DB2IMAGE or DB2VIDEO)
Column name or host variable that contains the handle of the image or video.

new_thumbnail (LONG VARCHAR FOR BIT DATA)
Source content for update of thumbnail

Return values (data type)

For retrieval, the content of the retrieved thumbnail (LONG VARCHAR FOR BIT DATA) for update, the handle of the image or video (DB2IMAGE or DB2VIDEO).

Examples

Get the thumbnail of Anita Jones's image stored in the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
 struct{
       short len;
       char data [32000];
         }hvThumbnail;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT THUMBNAIL(PICTURE)
     INTO :hvThumbnail
     FROM EMPLOYEE
     WHERE NAME = 'Anita Jones';
```

Update the thumbnail associated with Anita Jones's video in the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
     struct {
       short len;
       char data[10000];
     }hvThumbnail;
EXEC SQL END DECLARE SECTION;

/*  Create thumbnail and    */
/*  store in hvThumbnail    */

EXEC SQL UPDATE EMPLOYEE
   SET VIDEO=THUMBNAIL(
               VIDEO,
               :hvThumbnail)
   WHERE NAME='Anita Jones';
```

TicksPerQNote

Image	Audio	Video
	X	

Returns the clock speed of a recorded MIDI audio, in ticks per quarter note.

Include file

dmbaudio.h

Syntax

Parameters (data type)

handle (DB2AUDIO)
Column name or host variable that contains the handle of the audio.

Return values (data type)

Number of clock ticks per quarter note of MIDI audio (SMALLINT). Returns a null value for audio in other formats.

Examples

Get the file names of all MIDI audios in the sound column of the employee table that were recorded at speeds higher than 200 clock ticks per quarter note:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
       INTO :hvAud_fname
       FROM EMPLOYEE
       WHERE FORMAT(SOUND)='MIDI'
       AND TICKSPERQNOTE(SOUND)>200;
```

TicksPerSec

Image	Audio	Video
	X	

Returns the clock speed of a recorded MIDI audio, in ticks per second.

Include file

dmbaudio.h

Syntax

Parameters (data type)

handle (DB2AUDIO)
Column name or host variable that contains the handle of the audio.

Return values (data type)

Number of clock ticks per second of MIDI audio (SMALLINT). Returns a null value for audio in other formats.

Examples

Get the file names of all MIDI audios in the sound column of the employee table that were recorded at speeds less than 50 clock ticks per second:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvAud_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
      INTO :hvAud_fname
      FROM EMPLOYEE
      WHERE FORMAT(SOUND)='MIDI'
      AND TICKSPERSEC(SOUND)<50;
```

Updater

Image	Audio	Video
X	X	X

Returns the user ID of the person who last updated an image, audio, or video in a database table.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

Return values (data type)

User ID of person who last updated the image, audio, or video (CHAR(8))

Examples

Get the user ID of the person who last updated the video stored in the video column of the employee table for Robert Smith:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvUpdater[30];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT UPDATER(VIDEO)
     INTO :hvUpdater
     FROM EMPLOYEE
     WHERE NAME='rsmith';
```

UpdateTime

Image	Audio	Video
X	X	X

Returns a timestamp that indicates when an image, audio, or video in a database table was last updated.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE, DB2AUDIO, or DB2VIDEO)
Column name or host variable that contains the handle of the image, audio, or video.

Return values (data type)

Timestamp when image, audio, or video was last updated (TIMESTAMP)

Examples

Get the names of files for images in the picture column of the employee table that were updated in the last 2 days:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvImg_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE(CURRENT TIMESTAMP -
       UPDATETIME(PICTURE))< 2;
```

Width

Image	Audio	Video
X		X

Returns the width in pixels of an image or video frame.

Include file

image dmbimage.h

video dmbvideo.h

Syntax

Parameters (data type)

handle (DB2IMAGE or DB2VIDEO)
Column name or host variable that contains the handle of the image or video.

Return values (data type)

Width, in pixels (INTEGER)

Examples

Get the file name of all images in the picture column of the employee table that are narrower than 300 pixels:

```
EXEC SQL BEGIN DECLARE SECTION;
 char hvImg_fname[251];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
     INTO :hvImg_fname
     FROM EMPLOYEE
     WHERE WIDTH(PICTURE)<300;
```

Chapter 16. Application Programming Interfaces

This chapter gives reference information for the DB2 Extender administrative APIs. The APIs are listed in alphabetical order.

The following information is presented for each API:

- The extender that implements the API
- A brief description
- The authorization needed to use this API
- The library file for the API
- The include (header) file for the API
- The C syntax of the API call
- A description of the API parameters
- Values returned by the API
- Examples of use

DBaAdminGetInaccessibleFiles

Image	Audio	Video
	X	

Returns the names of inaccessible files that are referenced in audio columns of user tables. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaAdminGetInaccessibleFiles(
      char *qualifier,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of inaccessible files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

SQL_ERROR or other SQL return codes
Error returned from DB2.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

Examples

List all inaccessible files referenced in audio columns of tables owned by user ID rsmith:

```
#include <dmbaudio.h>
long idx;

rc = DBaAdminGetInaccessibleFiles("rsmith",
       &count, &filelist);
```

DBaAdminGetReferencedFiles

Image	Audio	Video
	X	

Returns the names of files that are referenced in audio columns of user tables. If a file is inaccessible (for example, its file name cannot be resolved using environment variable specifications), the file name is preceded with an asterisk. This API does not use the FILENAME field of the FILEREF data structure, and therefore sets it to NULL. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaAdminGetReferencedFiles(
      char *qualifier,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

Examples

List all files that are referenced in audio columns in tables owned by ajones:

```
#include <dmbaudio.h>
long idx;

rc = DBaAdminGetReferencedFiles("ajones",
       &count, &fileList);
```

DBaAdminIsFileReferenced

Image	Audio	Video
	X	

Returns a list of audio column entries in user tables that reference a specified file. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaAdminIsFileReferenced(
      char *qualifier,
      char *fileName,
      long *count,
      FILEREF *(*tableList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

fileName (in)
the name of the referenced file.

count (out)
The number of entries in the output list.

tableList (out)
A list of table entries that reference the specified file.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

Examples

List the entries in audio columns in all tables in the current database that reference file /audios/asmith.wav:

```
#include <dmbaudio.h>
long idx;

rc = DBaAdminIsFileReferenced(NULL,
       "/audios/asmith.wav",
       &count, &tableList);
```

DBaAdminReorgMetadata

Image	Audio	Video
	X	

"Cleans up" audio-related metadata tables, for example:

- Reclaims space that is no longer used in audio metadata tables
- Deletes references in audio metadata tables to audio files that no longer exist

The application must be connected to a database before calling this API.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaAdminReorgMetadata(
      char *qualifier
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are cleaned up. If a null value is specified, all tables in the current database are cleaned up.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

Examples

Clean up the metadata tables for audio columns in tables owned by user ID rsmith:

```
#include <dmbaudio.h>

rc = DBaAdminReorgMetadata("rsmith");
```

DBaDisableColumn

Image	Audio	Video
	X	

Disables a column for audio (DB2Audio data) so that it cannot hold audio data. The contents of the column entries are set to NULL, and the metadata associated with this column is dropped. All the triggers defined by the audio extender for this column are also dropped. New rows can be inserted into the table that contains the disabled column, and the new rows can include data defined with type DB2Audio, but there is no metadata (in the administrative support tables) associated with the new rows. The application must be connected to a database before calling this API. It is recommended that after calling this API you issue an SQL COMMIT statement.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaDisableColumn(
      char *tableName,
      char *colName,
      );
```

Parameters

tableName (in)
The name of the table that contains the audio column.

colName (in)
The name of the audio column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the sound column in the employee table for audio (DB2Audio data):

```
#include <dmbaudio.h>

rc = DBaDisableColumn("employee", "sound");
```

DBaDisableDatabase

Image	Audio	Video
	X	

Disables a database for audio (DB2Audio data) so that it cannot hold audio data. All tables in the database defined for DB2Audio are also disabled. The metadata and UDFs defined by the Audio Extender for the database are dropped. New rows can be inserted into tables in the database that are defined with type DB2Audio, but there is no metadata (in the administrative support tables) associated with the new rows. It is recommended that after calling this API you issue an SQL COMMIT statement.

Authorization

DBADM, SYSADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaDisableDatabase(
     );
```

Parameters

DBaDisableDatabase has no parameters.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the current database for audio (DB2Audio data):

```
#include <dmbaudio.h>

rc = DBaDisableDatabase();
```

DBaDisableTable

Image	Audio	Video
	X	

Disables a table for audio (DB2Audio data) so that it cannot hold audio data. All columns in the table defined for DB2Audio are also disabled. Some of the metadata defined by the Audio Extender for the table is dropped. New rows can be inserted into tables that are defined with type DB2Audio, but there is no metadata (in the administrative support tables) associated with the new rows. The application must be connected to a database before calling this API. It is recommended that after calling this API you issue an SQL COMMIT statement.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaDisableTable(
      char *tableName
      );
```

Parameters

tableName (in)
The name of the table that contains an audio column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the employee table for audio (DB2Audio data):

```
#include <dmbaudio.h>

rc = DBaDisableTable("employee");
```

DBaEnableColumn

Image	Audio	Video
	X	

Enables a column for audio (DB2Audio data). The API defines and manages relationships between this column and the metadata tables. Before calling this API, the application must be connected to a database. It is recommended that after calling this API you issue an SQL COMMIT statement.

Authorization

Control, Alter, SYSADM, DBADM

Use privilege is also required on table spaces and buffer pools specified in the API parameters.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaEnableColumn(
      char *tableName,
      char *colName,
      );
```

Parameters

tableName (in)
The name of the table that contains the audio column.

colName (in)
The name of the audio column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
Column is already enabled.

MMDB_RC_WRONG_SIGNATURE
Data type for the specified column is incorrect. User-defined data type MMDBSYS.DB2AUDIO is expected.

MMDB_RC_COLUMN_DOESNOT_EXIST
Column is not defined in the specified table.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NOT_ENABLED
Database or table is not enabled.

Examples

Enable the sound column in the employee table for audio (DB2Audio data):

```
#include <dmbaudio.h>

rc = DBaEnableColumn("employee", "sound");
```

DBaEnableDatabase

Image	Audio	Video
	X	

Enables a database for audio (DB2Audio data). This API is called once per database. It defines a DB2 user-defined type, DB2Audio, to the database manager. It also creates all UDFs that manipulate DB2Audio data. It is recommended that after calling this API you issue an SQL COMMIT statement.

Authorization

DBADM, SYSADM, SYSCTRL

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaEnableDatabase(
      char *tableSpace
      );
```

Parameters

tableSpace (in)

The name of the table space, which is a collection of containers into which administrative tables are stored. The table space specification has three parts as follows: *datats, indexts, longts*, where *datats* is the table space in which metadata tables are created; *indexts* is the table space in which indexes on the metadata tables are created; and *longts* is the table space in which values of long columns in the metadata tables (such as those that contain LONG VARCHAR and LOB data types) are stored. If you provide a null value for any part of the table space specification, the default table space for that part is used.

EEE Only: The tablespaces specified when enabling a database for an extender should be defined on a nodegroup that includes all the nodes in the partitioned database system.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
The database is already enabled.

MMDB_RC_API_NOT_SUPPORTED_FOR_SERVER
The server connected to does not support this command.

MMDB_WARN_NOT_ALL_NODES
Tablespace specified does not include all nodes for the extender. **(EEE Only)**

MMDB_RC_NOT_SAME_NODEGROUP
Tablespaces specified are not in the same nodegroup. **(EEE Only)**

Examples

Enable the current database for audio (DB2Audio data) in the table space MYTS. Use defaults for the index and long table spaces:

```
#include <dmbaudio.h>

rc = DBaEnableDatabase("myts,,");
```

Enable the current database for audio (DB2Audio data); use default table spaces:

```
#include <dmbaudio.h>

rc = DBaEnableDatabase(NULL);
```

DBaEnableTable

Image	Audio	Video
	X	

Enables a table for audio (DB2Audio data). This API is called once per table. It creates metadata tables to store and manage attributes for audio columns in a table. To avoid the possibility of locking, the application should commit transactions before calling this API. Before calling this API, the application must be connected to a database. It is recommended that after calling this API you issue an SQL COMMIT statement.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaEnableTable(
      char *tableSpace,
      char *tableName
      );
```

Parameters

tableSpace (in)

The name of the table space, which is a collection of containers into which administrative tables are stored. The table space specification has three parts as follows: *datats, indexts, longts*, where *datats* is the table space in which metadata tables are created; *indexts* is the table space in which indexes on the metadata tables are created; and *longts* is the table space in which values of long columns in the metadata tables (such as those that contain LONG VARCHAR and LOB data types) are stored. If you provide a null value for any part of the table space specification, the default table space for that part is used.

If you provide a null value for any part of the table space specification, the default table space for that part is used.

EEE Only: The tablespace specified should be in the same nodegroup as the user table.

tableName (in)
The name of the table that will contain an audio column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
Table is already enabled.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_TABLE_DOESNOT_EXIST
Table does not exist.

MMDB_RC_TABLESPACE_NOT_SAME_NODEGROUP
Tablespace specified is not in the same nodegroup as the user table. **(EEE Only)**

Examples

Enable the employee table for audio (DB2Audio data) in the table space MYTS. Use defaults for the index and long table spaces:

```
#include <dmbaudio.h>

rc = DBaEnableTable("myts,,",
       "employee");
```

Enable the employee table for audio (DB2Audio data). Use default table spaces:

```
#include <dmbaudio.h>

rc = DBaEnableTable(NULL,
       "employee");
```

DBaGetError

Image	Audio	Video
	X	

Returns a description of the last error. Call this API after any other API returns an error code.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaGetError(
      SQLINTEGER *sqlcode,
      char *errorMsgText
      );
```

Parameters

sqlcode (out)
The generic SQL error code.

errorMsgText (out)
The SQL error message text.

Error codes

MMDB_SUCCESS
API call processed successfully.

Examples

Get the last error, storing the SQL error code in `errCode` and the message text in `errMsg`:

```
#include <dmbaudio.h>

rc = DBaGetError(&errCode, &errMsg);
```

DBaGetInaccessibleFiles

Image	Audio	Video
	X	

Returns the names of inaccessible files that are referenced in audio columns of user tables. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SELECT privilege on enabled audio columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaGetInaccessibleFiles(
      char *tableName,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

tableName (in)
A qualified, unqualified, or null table name. If a table name is specified, that table is searched for references to inaccessible files. If a null value is specified, all tables with the specified qualifier are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of inaccessible files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all inaccessible files referenced in audio columns in the employee table:

```
long idx;
#include <dmbaudio.h>

rc = DBaGetInaccessibleFiles("employee",
       &count, &filelist);
```

DBaGetReferencedFiles

Image	Audio	Video
	X	

Returns the names of files that are referenced in audio columns of user tables. If a file is inaccessible (for example, its file name cannot be resolved using environment variable specifications), the file name is preceded with an asterisk. This API does not use the FILENAME field of the FILEREF data structure, and therefore sets it to NULL. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure.

Authorization

SELECT privilege on enabled audio columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaGetReferencedFiles(
      char *tableName,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

tableName (in)
A qualified, unqualified , or null table name. If a table name is specified, that table is searched for references to files. If a null value is specified, all tables owned by the current user ID database are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all files that are referenced in audio columns in the employee table:

```
#include <dmbaudio.h>
long idx;

rc = DBaGetReferencedFiles("employee",
       &count, &filelist);
```

DBaIsColumnEnabled

Image	Audio	Video
	X	

Determines whether a column has been enabled for audio (DB2Audio data). The application must be connected to a database before calling this API.

Authorization

SYSADM, DBADM, table owner, or SELECT privilege on the user table

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaIsColumnEnabled(
      char *tableName,
      char *colName,
      short *status
      );
```

Parameters

tableName (in)
A qualified or unqualified table name.

colName (in)
The name of a column.

status (out)
Indicates whether the column is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

0 MMDB_IS_NOT_ENABLED

-1 MMDB_INVALID_DATATYPE

DBaIsColumnEnabled

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the sound column in the employee table is enabled for audio:

```
#include <dmbaudio.h>

rc = DBaIsColumnEnabled("employee",
       "sound", &status);
```

DBaIsDatabaseEnabled

Image	Audio	Video
	X	

Determines whether a database has been enabled for audio (DB2Audio data). The application must be connected to a database before calling this API.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaIsDatabaseEnabled(
      short *status
      );
```

Parameters

status (out)
Indicates whether the database is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

0 MMDB_IS_NOT_ENABLED

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the personnl database is enabled for audio:

```
#include <dmbaudio.h>

rc = DBaIsDatabaseEnabled(&status);
```

DBaIsFileReferenced

Image	Audio	Video
	X	

Returns a list of table entries that reference a specified file. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SELECT privilege on enabled audio columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaIsFileReferenced(
      char *tableName,
      char *fileName,
      long *count,
      FILEREF *(*tableList)
      );
```

Parameters

tableName (in)
A qualified, unqualified , or null table name. If a table name is specified, that table is searched for references to the specified file. If a null value is specified, all tables owned by the current user ID are searched.

fileName (in)
The name of the referenced file.

count (out)
The number of entries in the output list.

tableList (out)
A list of table entries that reference the specified file.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List the entries in audio columns of the employee table that reference file /audios/ajones.wav:

```
#include <dmbaudio.h>
long idx;

rc = DBaIsFileReferenced(NULL,
       "/audios/ajones.wav",
       &count, &tableList);
```

DBaIsTableEnabled

Image	Audio	Video
	X	

Determines whether a table has been enabled for audio (DB2Audio data). The application must be connected to a database before calling this API.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaIsTableEnabled(
      char *tableName,
      short *status
      );
```

Parameters

tableName (in)
A table name.

status (out)
Indicates whether the table is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

0 MMDB_IS_NOT_ENABLED

-1 MMDB_INVALID_DATATYPE

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the employee table is enabled for audio (DB2Audio data):

```
#include <dmbaudio.h>

rc = DBaIsTableEnabled("employee", &status);
```

DBaPlay

Image	Audio	Video
	X	

Opens the audio player on the client and plays an audio clip. The clip can be stored in an audio column or an external file:

- If the audio clip is stored in an external file, you can pass either the name of the file or the audio handle to this API. The API uses the client environment variable DB2AUDIOPATH to resolve the file location. The file must be accessible from the client workstation.
- If the audio clip is stored in a column, you must pass the audio handle to the API. The application must be connected to the database and have read access to the table in which the audio clip is stored.

If the audio is stored in a column, the extender creates a temporary file and copies the content of the object from the column to the file. The extender might also create a temporary file if the audio is stored in an external file and its relative filename cannot be resolved using the values in environment variables, or if the file is not accessible on the client machine. The temporary file is created in the directory specified in the DB2AUDIOTEMP environment variable. The extender then plays the audio from the temporary file.

Authorization

Select authority on the user table, if playing an audio clip from a column.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

Play an audio stored in a column

```
long DBaPlay(
      char *playerName,
      MMDB_PLAY_HANDLE,
      DB2Audio *audioHandle,
      waitFlag
      );
```

DBaPlay

Syntax

Play an audio stored as a file

```
long DBaPlay(
      char *playerName,
      MMDB_PLAY_FILE,
      char *fileName,
      waitFlag
      );
```

Parameters

playerName (in)
The name of the audio player. If set to NULL, the default audio player specified by the DB2AUDIOPLAYER environment variable is used.

MMDB_PLAY_HANDLE (in)
A constant that indicates the audio is stored as a BLOB.

MMDB_PLAY_FILE (in)
A constant that indicates the audio is stored as a file that is accessible from the client.

audioHandle (in)
The handle of the audio. This parameter must be passed when you play an audio clip in a column. If the audio handle represents an external file, the client environment variable DB2VIDEOPATH is used to resolve the file location.

fileName (in)
The name of the file that contains the audio.

waitFlag (in)
A constant that indicates whether your program waits for the user to close the player before continuing. MMDB_PLAY_WAIT runs the player in the same thread as your application. MMDB_PLAY_NO_WAIT runs the player in a separate thread.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Play the audio identified by the audioHandle. Run the default player in the same thread as the application:

```
#include <dmbaudio.h>

rc = DBaPlay(NULL, MMDB_PLAY_HANDLE,
       audioHandle, MMDB_PLAY_WAIT);
```

DBaPrepareAttrs

Image	Audio	Video
	X	

Prepares user-supplied audio attributes. This API is used when an audio object with user-supplied attributes is stored or updated. The UDF code that runs on the server always expects data in “big endian”format, a format used by most UNIX platforms. If an audio object is stored or updated in “little endian” format, that is, from a non-UNIX client, the DBaPrepare API must be used before the store or update request is made.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudio.lib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
void DBaPrepareAttrs(
      MMDBAudioAttrs *audAttr
      );
```

Parameters

audAttr (in)
The user-supplied attributes of the audio.

Examples

Prepare user-supplied audio attributes:

```
#include <dmbaudio.h>

DBaPrepareAttrs(&imgattr);
```

DBaReorgMetadata

Image	Audio	Video
	X	

"Cleans up" audio-related metadata tables, for example:

- Reclaims space that is no longer used in audio metadata tables
- Deletes references in audio metadata tables to audio files that no longer exist

The application must be connected to a database before calling this API.

Authorization

Alter, Control, SYSADM, SYSCTRL, SYSMAINT, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbaudiolib	libdmbaudio.a (AIX) libdmbaudio.sl (HP-UX) libdmbaudio.so (Solaris)

Include file

dmbaudio.h

Syntax

```
long DBaReorgMetadata(
      char *tableName
      );
```

Parameters

tableName (in)
A qualified, unqualified, or null table name. If a table name is specified, cleanup is performed for audio metadata tables associated with the specified user table. If a null value is specified, metadata tables for audio columns in all tables owned by the current user ID are cleaned up.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Clean up the metadata tables for audio columns in the employee table:

```
#include <dmbaudio.h>

rc = DBaReorgMetadata("employee");
```

DBiAdminGetInaccessibleFiles

Image	Audio	Video
X		

Returns the names of inaccessible files that are referenced in image columns of user tables. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiAdminGetInaccessibleFiles(
      char *qualifier,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of inaccessible files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all inaccessible files referenced in image columns of tables owned by user ID rjones:

```
#include <dmbimage.h>
long idx;

rc = DBiAdminGetInaccessibleFiles
       ("rjones", &count, &filelist);
```

DBiAdminGetReferencedFiles

Image	Audio	Video
X		

Returns the names of files that are referenced in image columns of user tables. If a file is inaccessible (for example, its file name cannot be resolved using environment variable specifications), the file name is preceded with an asterisk. This API does not use the FILENAME field of the FILEREF data structure, and therefore sets it to NULL. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiAdminGetReferencedFiles(
      char *qualifier,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of files that are referenced in the table.

DBiAdminGetReferencedFiles

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all files that are referenced in image columns in tables owned by ajones:

```
#include <dmbimage.h>
long idx;

rc = DBiAdminGetReferencedFiles("ajones",
       &count, &filelist);
```

DBiAdminIsFileReferenced

Image	Audio	Video
X		

Returns a list of image column entries in user tables that reference a specified file. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiAdminIsFileReferenced(
      char *qualifier,
      char *fileName,
      long *count,
      FILEREF *(*tableList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

fileName (in)
The name of the referenced file.

count (out)
The number of entries in the output list.

tableList (out)
A list of table entries that reference the specified file.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List the entries in image columns in all tables in the current database that reference file /images/asmith.bmp:

```
#include <dmbimage.h>
long idx;

rc = DBiAdminIsFileReferenced(NULL,
       "/images/asmith.bmp",
       &count, &tableList);
```

DBiAdminReorgMetadata

Image	Audio	Video
X		

"Cleans up" image-related metadata tables:

- Reclaims space that is no longer used in image metadata tables
- Deletes references in image metadata tables to image files that no longer exist

The application must be connected to a database before calling this API.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiAdminReorgMetadata(
      char *qualifier
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are cleaned up. If a null value is specified, all tables in the current database are cleaned up.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

Examples

Clean up the metadata tables for image columns in tables owned by user ID rsmith:

```
#include <dmbimage.h>

rc = DBiAdminReorgMetadata("rsmith");
```

DBiBrowse

Image	Audio	Video
X		

Opens the image browser on the client and displays an image. The image can be stored in an image column or an external file:

- If the image is stored in an external file, you can pass either the name of the file or the image handle to this API. The API uses the client environment variable DB2IMAGEPATH to resolve the file location. The file must be accessible from the client workstation.
- If the image is stored in a column, you must pass the image handle to the API. The application must be connected to the database and have read access to the table in which the image is stored.

If the browser can not directly access the image, the extender creates a temporary file in the directory specified in the DB2IMAGETEMP environment variable. The extender then displays the image from the temporary file.

Authorization

Select authority on the user table, if browsing an image from a column.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

Browse an image stored in a column

```
long DBiBrowse(
      char *browserName,
      MMDB_PLAY_HANDLE,
      DB2Image *imageHandle,
      waitFlag
      );
```

Syntax

Browse an image stored as a file

DBiBrowse

```
long DBiBrowse(
      char *browserName,
      MMDB_PLAY_FILE,
      char *fileName,
      waitFlag
      );
```

Parameters

browserName (in)
The name of the image browser. If set to NULL, the default image browser specified by the DB2IMAGEBROWSER environment variable is used.

MMDB_PLAY_HANDLE (in)
A constant that indicates the image is stored as a BLOB.

MMDB_PLAY_FILE (in)
A constant that indicates the image is stored as a file that is accessible from the client.

imageHandle (in)
The handle of the image. This parameter must be passed when you browse an image in a column. If the image handle represents an external file, the client environment variable DB2IMAGEPATH is used to resolve the file location.

fileName (in)
The name of the file that contains the image.

waitFlag (in)
A constant that indicates whether your program waits for the user to close the browser before continuing. MMDB_PLAY_WAIT runs the browser in the same thread as your application.
MMDB_PLAY_NO_WAIT runs the browser in a separate thread.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Display the image identified by the imageHandle. Run the default browser in the same thread as the application:

```
#include <dmbimage.h>

rc = DBiBrowse(NULL, MMDB_PLAY_HANDLE,
       imageHandle, MMDB_PLAY_WAIT);
```

DBiDisableColumn

Image	Audio	Video
X		

Disables a column for images (DB2Image data) so that it cannot hold image data. The contents of the column entries are set to NULL, and the metadata associated with this column is dropped. All the triggers defined by the image extender for this column are also dropped. New rows can be inserted into the table that contains the disabled column, and the new rows can include data defined with type DB2Image, but there is no metadata (in the administrative support tables) associated with the new rows. The application must be connected to a database before calling this API.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiDisableColumn(
      char *tableName,
      char *colName,
      );
```

Parameters

tableName (in)
The name of the table that contains the image column.

colName (in)
The name of the image column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the picture column in the employee table for images (DB2Image data):

```
#include <dmbimage.h>

rc = DBiDisableColumn("employee",
       "picture");
```

DBiDisableDatabase

Image	Audio	Video
X		

Disables a database for images (DB2Image data) so that it cannot hold image data. All tables in the database defined for DB2Image are also disabled. The metadata and UDFs defined by the Image Extender for the database are dropped. New rows can be inserted into tables in the database that are defined with type DB2Image, but there is no metadata (in the administrative support tables) associated with the new rows.

Authorization

DBADM, SYSADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiDisableDatabase(
     );
```

Parameters

DBiDisableDatabase has no parameters.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the current database for images (DB2Image data):

```
#include <dmbimage.h>

rc = DBiDisableDatabase();
```

DBiDisableTable

Image	Audio	Video
X		

Disables a table for images (DB2Image data) so that it cannot hold image data. All columns in the table defined for DB2Image are also disabled. Some of the metadata defined by the Image Extender for the table is dropped. New rows can be inserted into tables that are defined with type DB2Image, but there is no metadata (in the administrative support tables) associated with the new rows. The application must be connected to a database before calling this API.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiDisableTable(
      char *tableName
      );
```

Parameters

tableName (in)
The name of the table that contains an image column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the employee table for images (DB2Image data):

```
#include <dmbimage.h>

rc = DBiDisableTable("employee");
```

DBiEnableColumn

Image	Audio	Video
X		

Enables a column for images (DB2Image data). The API defines and manages relationships between this column and the metadata tables. Before calling this API, the application must be connected to a database and the user table must be committed.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiEnableColumn(
      char *tableName,
      char *colName,
      );
```

Parameters

tableName (in)
The name of the table that contains the image column.

colName (in)
The name of the image column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
Column is already enabled.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_WRONG_SIGNATURE
Datatype for the specified column is incorrect. User-defined datatype MMDBSYS.DB2IMAGE is expected.

MMDB_RC_COLUMN_DOESNOT_EXIST
Column is not defined in the specified table.

MMDB_RC_NOT_ENABLED
Database or table is not enabled.

Examples

Enable the picture column in the employee table for images:

```
#include <dmbimage.h>

rc = DBiEnableColumn("employee",
       "picture");
```

DBiEnableDatabase

Image	Audio	Video
X		

Enables a database for images (DB2Image data). This API is called once per database. It defines a DB2 user-defined type, DB2Image, to the database manager. It also creates all UDFs that manipulate DB2Image data.

Authorization

DBADM, SYSADM, SYSCTRL

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiEnableDatabase(
      char *tableSpace
      );
```

Parameters

tableSpace (in)

The name of the table space, which is a collection of containers into which administrative tables are stored. The table space specification has three parts as follows: *datats, indexts, longts*, where *datats* is the table space in which metadata tables are created; *indexts* is the table space in which indexes on the metadata tables are created; and *longts* is the table space in which values of long columns in the metadata tables (such as those that contain LONG VARCHAR and LOB data types) are stored. If you provide a null value for any part of the table space specification, the default table space for that part is used.

EEE Only: The tablespaces specified when enabling a database for an extender should be defined on a nodegroup that includes all the nodes in the partitioned database system.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
The database is already enabled.

MMDB_RC_API_NOT_SUPPORTED_FOR_SERVER
The server connected to does not support this command.

MMDB_WARN_NOT_ALL_NODES
Tablespace specified does not include all nodes for the extender. **(EEE Only)**

MMDB_RC_NOT_SAME_NODEGROUP
Tablespaces specified are not in the same nodegroup. **(EEE Only)**

Examples

Enable the current database for images (DB2Image data) in the table space named MYTS. Use defaults for the index and long table spaces:

```
#include <dmbimage.h>

rc = DBiEnableDatabase("myts,,");
```

Enable the current database for images (DB2Image data). Use default table spaces:

```
#include <dmbimage.h>

rc = DBiEnableDatabase(NULL);
```

DBiEnableTable

Image	Audio	Video
X		

Enables a table for images (DB2Image data). This API is called once per table. It creates metadata tables to store and manage attributes for image columns in a table. To avoid the possibility of locking, the application should commit transactions before calling this API. Before calling this API, the application must be connected to a database.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiEnableTable(
      char *tableSpace,
      char *tableName
      );
```

Parameters

tableSpace (in)

The name of the table space, which is a collection of containers into which administrative tables are stored. The table space specification has three parts as follows: *datats, indexts, longts*, where *datats* is the table space in which metadata tables are created; *indexts* is the table space in which indexes on the metadata tables are created; and *longts* is the table space in which values of long columns in the metadata tables (such as those that contain LONG VARCHAR and LOB data types) are stored. If you provide a null value for any part of the table space specification, the default table space for that part is used.

If you provide a null value for any part of the table space specification, the default table space for that part is used.

EEE Only: The tablespace specified should be in the same nodegroup as the user table.

tableName (in)
The name of the table that will contain an image column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
Table is already enabled.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_TABLE_DOESNOT_EXIST
Table does not exist.

MMDB_RC_TABLESPACE_NOT_SAME_NODEGROUP
Tablespace specified is not in the same nodegroup as the user table. **(EEE Only)**

Examples

Enable the employee table for images (DB2Image data) in the table space MYTS. Use defaults for the index and long table spaces:

```
#include <dmbimage.h>

rc = DBiEnableTable("myts,,",
       "employee");
```

Enable the employee table for images (DB2Image data). Use default table spaces:

```
#include <dmbimage.h>

rc = DBiEnableTable(NULL,
       "employee");
```

DBiGetError

Image	Audio	Video
X		

Returns a description of the last error. Call this API after any other API returns an error code.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiGetError(
      SQLINTEGER *sqlcode,
      char *errorMsgText
      );
```

Parameters

sqlcode (out)
The generic SQL error code.

errorMsgText (out)
The SQL error message text.

Error codes

MMDB_SUCCESS
API call processed successfully.

Examples

Get the last error, storing the SQL error code in `errCode` and the message text in `errMsg`:

```
#include <dmbimage.h>

rc = DBiGetError(&errCode, &errMsg);
```

DBiGetInaccessibleFiles

Image	Audio	Video
X		

Returns the names of inaccessible files that are referenced in image columns of user tables. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SELECT privilege on enabled image columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiGetInaccessibleFiles(
      char *tableName,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

tableName (in)
A qualified, unqualified, or null table name. If a table name is specified, that table is searched for references to inaccessible files. If a null value is specified, all tables with the specified qualifier are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of inaccessible files that are referenced in the table.

DBiGetInaccessibleFiles

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all inaccessible files referenced in image columns in the employee table:

```
A#include <dmbimage.h>
long idx;

rc = DBiGetInaccessibleFiles("employee",
       &count, &filelist);
```

DBiGetReferencedFiles

Image	Audio	Video
X		

Returns the names of files that are referenced in image columns of user tables. If a file is inaccessible (for example, its file name cannot be resolved using environment variable specifications), the file name is preceded with an asterisk. This API does not use the FILENAME field of the FILEREF data structure, and therefore sets it to NULL. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure.

Authorization

SELECT privilege on enabled image columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiGetReferencedFiles(
      char *tableName,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

tableName (in)
A qualified, unqualified , or null table name. If a table name is specified, that table is searched for references to files. If a null value is specified, all tables owned by the current user ID are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all files that are referenced in image columns in the employee table:

```
#include <dmbimage.h>
long idx;

rc = DBiGetReferencedFiles("employee",
       &count, &filelist);
```

DBiIsColumnEnabled

Image	Audio	Video
X		

Determines whether a column has been enabled for images (DB2Image data). The application must be connected to a database before calling this API.

Authorization

SYSADM, DBADM, table owner, or SELECT privilege on the user table

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiIsColumnEnabled(
      char *tableName,
      char *colName,
      short *status
      );
```

Parameters

tableName (in)
A qualified or unqualified table name.

colName (in)
The name of a column.

status (out)
Indicates whether the column is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

0 MMDB_IS_NOT_ENABLED

-1 MMDB_INVALID_DATATYPE

DBiIsColumnEnabled

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
Column is already enabled.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the picture column in the employee table is enabled for images:

```
#include <dmbimage.h>

rc = DBiIsColumnEnabled("employee",
       "picture", &status);
```

DBiIsDatabaseEnabled

Image	Audio	Video
X		

Determines whether a database has been enabled for images (DB2Image data). The application must be connected to a database before calling this API.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiIsDatabaseEnabled(
      short *status
      );
```

Parameters

status (out)
Indicates whether the database is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

0 MMDB_IS_NOT_ENABLED

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the personnl database is enabled for images:

```
#include <dmbimage.h>

rc = DBiIsDatabaseEnabled(&status);
```

DBiIsFileReferenced

Image	Audio	Video
X		

Returns a list of table entries in image columns that reference a specified file. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SELECT privilege on enabled image columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiIsFileReferenced(
      char *tableName,
      char *fileName,
      long *count,
      FILEREF *(*tableList)
      );
```

Parameters

tableName (in)
A qualified, unqualified , or null table name. If a table name is specified, that table is searched for references to the specified file. If a null value is specified, all tables owned by the current user ID are searched.

fileName (in)
The name of the referenced file.

count (out)
The number of entries in the output list

tableList (out)
A list of table entries that reference the specified file

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List the entries in image columns of the employee table that reference file /images/ajones.bmp:

```
#include <dmbimage.h>
long idx;

rc = DBiIsFileReferenced(NULL,
       "/images/ajones.bmp",
       &count, &tableList);
```

DBiIsTableEnabled

Image	Audio	Video
X		

Determines whether a table has been enabled for images (DB2Image data). The application must be connected to a database before calling this API.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiIsTableEnabled(
      char *tableName,
      short *status
      );
```

Parameters

tableName (in)
A table name.

status (out)
Indicates whether the table is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the employee table is enabled for images:

```
#include <dmbimage.h>

rc = DBiIsTableEnabled("employee",
       &status);
```

DBiPrepareAttrs

Image	Audio	Video
X		

Prepares user-supplied image attributes. This API is used when an image object with user-supplied attributes is stored or updated. The UDF code that runs on the server always expects data in “big endian”format, a format used by most UNIX platforms. If an image object is stored or updated in “little endian” format, that is, from a non-UNIX client, the DBiPrepare API must be used before the store or update request is made.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
void DBiPrepareAttrs(
      MMDBImageAttrs *imgAttr
      );
```

Parameters

imgAttr (in)
The user-supplied attributes of the image.

Examples

Prepare user-supplied image attributes:

```
#include <dmbimage.h>

DBiPrepareAttrs(&imgattr);
```

DBiReorgMetadata

Image	Audio	Video
X		

"Cleans up" image-related metadata tables, for example:

- Reclaims space that is no longer used in image metadata tables
- Deletes references in image metadata tables to image files that no longer exist

The application must be connected to a database before calling this API.

Authorization

Alter, Control, SYSADM, SYSCTRL, SYSMAINT, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbimage.lib	libdmbimage.a (AIX) libdmbimage.sl (HP-UX) libdmbimage.so (Solaris)

Include file

dmbimage.h

Syntax

```
long DBiReorgMetadata(
      char *tableName
      );
```

Parameters

tableName (in)
A qualified, unqualified, or null table name. If a table name is specified, clean up is performed for image metadata tables associated with the specified user table. If a null value is specified, metadata tables for image columns in all tables owned by the current user ID are cleaned up.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Clean up the metadata tables for image columns in the employee table:

```
#include <dmbimage.h>

rc = DBiReorgMetadata("employee");
```

DBvAdminGetInaccessibleFiles

Image	Audio	Video
		X

Returns the names of inaccessible files that are referenced in video columns of user tables. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvAdminGetInaccessibleFiles(
      char *qualifier,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. (in) If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of inaccessible files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all inaccessible files referenced in video columns of tables owned by user ID rsmith:

```
#include <dmbvideo.h>
long idx;

rc = DBvAdminGetInaccessibleFiles
       ("rsmith", &count,
       &filelist);
```

DBvAdminGetReferencedFiles

Image	Audio	Video
		X

Returns the names of files that are referenced in video columns of user tables. If a file is inaccessible (for example, its file name cannot be resolved using environment variable specifications), the file name is preceded with an asterisk. This API does not use the FILENAME field of the FILEREF data structure, and therefore sets it to NULL. The application must be connected to a abase before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvAdminGetReferencedFiles(
      char *qualifier,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all files that are referenced in video columns in tables owned by ajones:

```
#include <dmbvideo.h>
long idx;

rc = DBvAdminGetReferencedFiles
       ("ajones", &count,
       &filelist);
```

DBvAdminIsFileReferenced

Image	Audio	Video
		X

Returns a list of video column entries in user tables that reference a specified file. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvAdminIsFileReferenced(
      char *qualifier,
      char *fileName,
      long *count,
      FILEREF *(*tableList)
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are searched. If a null value is specified, all tables in the current database are searched.

fileName (in)
The name of the referenced file.

count (out)
The number of entries in the output list.

tableList (out)
A list of table entries that reference the specified file.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List the entries in video columns in all tables in the current database that reference file /videos/asmith.mpg:

```
#include <dmbvideo.h>
long idx;

rc = DBvAdminIsFileReferenced(NULL,
       "/videos/asmith.mpg",
       &count, &tableList);
```

DBvAdminReorgMetadata

Image	Audio	Video
		X

"Cleans up" video-related metadata tables, for example:

- Reclaims space that is no longer used in video metadata tables
- Deletes references in video metadata tables to video files that no longer exist

The application must be connected to a database before calling this API.

Authorization

SYSADM, SYSCTRL, SYSMAINT

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvAdminReorgMetadata(
      char *qualifier
      );
```

Parameters

qualifier (in)
A valid user ID or a null value. If a user ID is specified, all tables with the specified qualifier are cleaned up. If a null value is specified, all tables in the current database are cleaned up.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_NO_AUTH
User does not have proper authority to call this API.

Examples

Clean up the metadata tables for video columns in tables owned by user ID rsmith:

```
#include <dmbvideo.h>

rc = DBvAdminReorgMetadata("rsmith");
```

DBvBuildStoryboardFile

Image	Audio	Video
		X

Creates a shot catalog file and builds entries in the file for all the shots in a video. The source video can be in a database or in a file. For each shot, the API stores the shot number, starting frame number, ending frame number, and information for at least one representative frame. Values in the DBvStoryboardCtrl data structure determine how many representative frames are identified for a shot. For shots whose length is below a threshold value in DBvStoryboardCtrl, the API identifies one representative frame. For shots whose length is between a lower and upper threshold value in DBvStoryboardCtrl, the API identifies two representative frames. For shots whose length is above the upper threshold value in DBvStoryboardCtrl, the API identifies three representative frames. The representative frame information includes its frame number and the name of the file that contains the frame content. This information can be used to display a storyboard, that is, a visual summary of a video.

Authorization

Insert, Control

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

For OS/2 and Windows: dmbshot.lib; for AIX, HP-UX, and Solaris: libdmbshot.a

Include file

dmbshot.h

Syntax

```
long DBvBuildStoryboardFile(
      char *fileName,
      DBvIOType *video,
      DBvShotControl *shotCtrl,
      DBvStoryBoardCtrl *sbCtrl
            );
```

Parameters

catalogName (in)
The pointer to the name of the shot catalog file.

video (in)
The pointer to the video structure.

shotCtrl (in)
The pointer to the shot control structure

sbCtrl (in)
The pointer to the storyboard control structure.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_INVALID_CATALOG
The catalog is not valid or does not exist.

Examples

Create a shot catalog file named hotshots and fill it with data for all the shots in a video:

```
#include <dmbshot.h>

rc = DBvBuildStoryboardFile("hotshots",
       video, &shotCtrl, &sbCtrl);
```

DBvBuildStoryboardTable

Image	Audio	Video
		X

Builds entries in a shot catalog for all the shots in a video. The source video can be in a database or in a file. The shot catalog is in a database. For each shot, the API stores the handle or file information for the source video. It also stores the shot number, starting frame number, ending frame number, and information for at least one representative frame. Values in the DBvStoryboardCtrl data structure determine how many representative frames are identified for a shot. For shots whose length is below a threshold value in DBvStoryboardCtrl, the API identifies one representative frame. For shots whose length is between a lower and upper threshold value in DBvStoryboardCtrl, the API identifies two representative frames. For shots whose length is above the upper threshold value in DBvStoryboardCtrl, the API identifies three representative frames. The representative frame information includes its frame number and frame data. The representative frame information stored in the shot catalog can be used to display a storyboard, that is, a visual summary of a video.

The application must be connected to a database before calling this API.

Authorization

Insert, Control

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvBuildStoryboardTable(
      char *catalogName,
      DBvIOType *video,
      DBvShotControl *shotCtrl,
      DBvStoryBoardCtrl *sbCtrl,
      SQLHDBC hdbc
      );
```

Parameters

catalogName (in)
The pointer to the name of the shot catalog.

video (in)
The pointer to the video structure.

shotCtrl (in)
The pointer to the shot control structure

sbCtrl (in)
The pointer to the storyboard control structure.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_INVALID_CATALOG
The catalog is not valid or does not exist.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Create entries in a shot catalog named hotshots for a video:

```
#include <dmbshot.h>

rc = DBvBuildStoryboardTable("hotshots",
       video, &shotCtrl, &sbCtrl, hdbc);
```

DBvClose

Image	Audio	Video
		X

Closes a video file that has been opened for scene change detection.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbmpeg.lib	libdmbmpeg.a (AIX) libdmbmpeg.sl (HP-UX) libdmbmeg.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvClose(
      DB2vIOType *video
      );
```

Parameters

video (in)
The pointer to the video structure.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_CANNOT_CLOSE
The video file could not be closed

Examples

Close a video file previously opened for video scene change detection:

```
#include <dmbshot.h>

rc = DBvClose(video);
```

DBvCreateIndex

Image	Audio	Video
		X

Creates an index for a video stored in a file. The index is used by the Video Extender to access shots and frames in a video. The index is stored in a flat file in the same directory as the source video file.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbmpeg.lib	libdmbmpeg.a (AIX) libdmbmpeg.sl (HP-UX) libdmbmeg.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvCreateIndex(
      char *fileName
      );
```

Parameters

fileName (in)
The pointer to a video file name.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_OPEN_VIDEO
The video file could not be opened for processing.

MMDB_RC_INDEX_FAIL
The index could not be built.

Examples

Create an index for the video in file \videos\ajones.mpg:

DBvCreateIndex

```
#include <dmbshot.h>

rc = DBvCreateIndex("\videos\ajones.mpg");
```

DBvCreateIndexFromVideo

Image	Audio	Video
		X

Creates an index for a video. The video must first be opened for shot detection. The index is used by the Video Extender to access shots and frames in a video. The index is stored in a flat file. The file name is stored in the DBvIOType data structure.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvCreateIndexFromVideo(
      DBvIOType *video
      );
```

Parameters

video (update)
The pointer to a video structure.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_OPEN_VIDEO
The video file could not be opened for processing.

MMDB_RC_INDEX_FAIL
The index could not be built.

Examples

Create an index for a video:

DBvCreateIndexFromVideo

```
#include <dmbshot.h>

rc = DBvCreateIndexFromVideo(video);
```

DBvCreateShotCatalog

Image	Audio	Video
		X

Creates a shot catalog, which is a set of tables that contains information about shots, such as frame numbers.

The application must be connected to a database that is enabled for both db2video and db2image.

Authorization

Create, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvCreateShotCatalog(
      char *catalogName,
      SQLHDBC hdbc
      );
```

Parameters

catalogName (in)
Name of the shot catalog to be created.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Create a shot catalog named hotshots:

```
#include <dmbshot.h>

rc = DBvCreateShotCatalog("hotshots", hdbc);
```

DBvDeleteShot

Image	Audio	Video
		X

Deletes a shot from a catalog.

Authorization

Insert, Control

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvDeleteShot(
      char *catalogName ,
      char *shotHandle,
      SQLHDBC hdbc
      );
```

Parameters

catalogName (in)
The name of the catalog.

shotHandle (in)
The shot handle.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_ACCESS
Caller does not have proper access authority.

MMDB_RC_NOT_CONNECTED
The application does not have valid connection to a database.

MMDB_RC_INVALID_CATALOG
The catalog is not valid or does not exist.

Examples

Delete a shot from the hotshots catalog, using the shot's handle:

```
#include <dmbshot.h>

rc = DBvDeleteShot("hotshots", shot,
       hdbc);
```

DBvDeleteShotCatalog

Image	Audio	Video
		X

Deletes a shot catalog.

Authorization

Control, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvDeleteShotCatalog(
      char *catalogName,
      SQLHDBC hdbc
      );
```

Parameters

catalogName (in)
The name of the shot catalog to be deleted.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_ACCESS
Caller does not have proper access authority.

MMDB_RC_NOT_CONNECTED
The application does not have valid connection to a database.

MMDB_RC_INVALID_CATALOG
The catalog is not valid or does not exist.

Delete the shot catalog hotshots:

Examples

```
#include <dmbshot.h>

rc = DBvDeleteShotCatalog("hotshots",
       hdbc);
```

DBvDetectShot

Image	Audio	Video
		X

Searches for the next shot in a video file. If a shot is detected, records the frame number and frame data of the first frame in the detected shot. You must examine the shotDetected flag to determine if a shot has been detected.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvDetectShot(
      DBvIOType *video,
      unsigned long *start_frame,
      char *shotDetected,
      DBvShotControl *shotCtrl,
      DBvShotType *shot,
            );
```

Parameters

video (update)
The pointer to the video structure.

start_frame (in/out)
The frame number used as the starting point for the search. On return, the parameter is updated with the position to start looking for the next shot.

shotDetected (out)
Shot detected flag: 1= frame detected, 0= no frame detected.

shotCtrl (in)
The pointer to the shot control data.

shot (out)
The pointer to the detected shot and shot data.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_EOF
End of file reached.

MMDB_NO_INDEX
The video index does not exist.

Examples

Search for the next shot in a video file starting from frame 1:

```
#include <dmbshot.h>

long start_frame=1;

rc = DBvDetectShot(video, start_frame&Detected,
       &shotCtrl, &shot);
```

DBvDisableColumn

Image	Audio	Video
		X

Disables a column for video (DB2Video data) so that it cannot hold video data. The contents of the column entries are set to NULL, and the metadata associated with this column is dropped. All the triggers defined by the video extender for this column are also dropped. New rows can be inserted into the table that contains the disabled column, and the new rows can include data defined with type DB2Video, but there is no metadata (in the administrative support tables) associated with the new rows. The application must be connected to a database before calling this API.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvDisableColumn(
      char *tableName,
      char *colName,
      );
```

Parameters

tableName (in)
The name of the table that contains the video column.

colName (in)
The name of the video column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the tv_ads column in the employee table for video (DB2Video data):

```
#include <dmbvideo.h>

rc = DBvDisableColumn("employee",
       "tv_ads");
```

DBvDisableDatabase

Image	Audio	Video
		X

Disables a database for video (DB2Video data) so that it cannot hold video data. All tables in the database defined for DB2Video are also disabled. The metadata and UDFs defined by the Video Extender for the database are dropped. New rows can be inserted into tables in the database that are defined with type DB2Video, but there is no metadata (in the administrative support tables) associated with the new rows.

Authorization

DBADM, SYSADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvDisableDatabase(
     );
```

Parameters

DBvDisableDatabase has no parameters.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the current database for video (DB2Video data):

DBvDisableDatabase

```
#include <dmbvideo.h>

rc = DBvDisableDatabase();
```

DBvDisableTable

Image	Audio	Video
		X

Disables a table for video (DB2Video data) so that it cannot hold video data. All columns in the table defined for DB2Video are also disabled. Some of the metadata defined by the Video Extender for the table is dropped. New rows can be inserted into tables that are defined with type DB2Video, but there is no metadata (in the administrative support tables) associated with the new rows. The application must be connected to a database before calling this API.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvDisableTable(
      char *tableName
      );
```

Parameters

tableName (in)
The name of the table that contains a video column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Disable the employee table for video (DB2Video data):

```
#include <dmbvideo.h>

rc = DBvDisableTable("employee");
```

DBvEnableColumn

Image	Audio	Video
		X

Enables a column for video (DB2Video data). The API defines and manages relationships between this column and the metadata tables. Before calling this API, the application must be connected to a database and the user table must be committed.

Authorization

Control, Alter, SYSADM, DBADM

Use privilege is also required on table spaces and buffer pools specified in the API parameters.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvEnableColumn(
      char *tableName,
      char *colName,
      );
```

Parameters

tableName (in)
The name of the table that contains the video column.

colName (in)
The name of the video column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
Column is already enabled.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_WRONG_SIGNATURE
Data type for the specified column is incorrect. User-defined data type MMDBSYS.DB2VIDEO is expected.

MMDB_RC_COLUMN_DOESNOT_EXIST
Column is not defined in the specified table.

MMDB_RC_NOT_ENABLED
Database or table is not enabled.

Examples

Enable the video column in the employee table for video:

```
#include <dmbvideo.h>

rc = DBvEnableColumn("employee",
       "video");
```

DBvEnableDatabase

Image	Audio	Video
		X

Enables a database for video (DB2Video data). This API is called once per database. It defines a DB2 user-defined type, DB2Video, to the database manager. It also creates all UDFs that manipulate DB2Video data.

Authorization

DBADM, SYSADM, SYSCTRL

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvEnableDatabase(
      char *tableSpace
      );
```

Parameters

tableSpace (in)
The name of the table space, which is a collection of containers into which administrative tables are stored. The table space specification has three parts as follows: *datats, indexts, longts*, where *datats* is the table space in which metadata tables are created; *indexts* is the table space in which indexes on the metadata tables are created; and *longts* is the table space in which values of long columns in the metadata tables (such as those that contain LONG VARCHAR and LOB data types) are stored. If you provide a null value for any part of the table space specification, the default table space for that part is used.

EEE Only: The tablespaces specified when enabling a database for an extender should be defined on a nodegroup that includes all the nodes in the partitioned database system.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
The database is already enabled.

MMDB_RC_API_NOT_SUPPORTED_FOR_SERVER
The server connected to does not support this command.

MMDB_WARN_NOT_ALL_NODES
Tablespace specified does not include all nodes for the extender. **(EEE Only)**

MMDB_RC_NOT_SAME_NODEGROUP
Tablespaces specified are not in the same nodegroup. **(EEE Only)**

Examples

Enable the current database for video (DB2Video data) in the table space named MYTS. Use defaults for the index and long table spaces:

```
#include <dmbvideo.h>

rc = DBvEnableDatabase("myts,,");
```

Enable the current database for video (DB2Video data). Use default table spaces:

```
#include <dmbvideo.h>

rc = DBvEnableDatabase(NULL);
```

DBvEnableTable

Image	Audio	Video
		X

Enables a table for video (DB2Video data). This API is called once per table. It creates metadata tables to store and manage attributes for video columns in a table. To avoid the possibility of locking, the application should commit transactions before calling this API. Before calling this API, the application must be connected to a database.

Authorization

Control, Alter, SYSADM, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvEnableTable(
      char *tableSpace,
      char *tableName
      );
```

Parameters

tableSpace (in)

The name of the table space, which is a collection of containers into which administrative tables are stored. The table space specification has three parts as follows: *datats, indexts, longts*, where *datats* is the table space in which metadata tables are created; *indexts* is the table space in which indexes on the metadata tables are created; and *longts* is the table space in which values of long columns in the metadata tables (such as those that contain LONG VARCHAR and LOB data types) are stored. If you provide a null value for any part of the table space specification, the default table space for that part is used.

If you provide a null value for any part of the table space specification, the default table space for that part is used.

EEE Only: The tablespace specified should be in the same nodegroup as the user table.

tableName (in)
The name of the table that will contain a video column.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_WARN_ALREADY_ENABLED
Table is already enabled.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_TABLE_DOESNOT_EXIST
Table does not exist.

MMDB_RC_TABLESPACE_NOT_SAME_NODEGROUP
Tablespace specified is not in the same nodegroup as the user table. **(EEE Only)**

Examples

Enable the employee table for video (DB2Video data) in the table space MYTS. Use defaults for the index and long table spaces:

```
#include <dmbvideo.h>

rc = DBvEnableTable("myts,,",
       "employee");
```

Enable the employee table for video (DB2Video data). Use default table spaces:

```
#include <dmbvideo.h>

rc = DBvEnableTable(NULL,
       "employee");
```

DBvFrameDataTo24BitRGB

Image	Audio	Video
		X

Converts a video frame from a YUV color-value format, such as MPEG, to a 24-bit RGB format. The user must allocate a target buffer before issuing the API call.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbmpeg.lib	libdmbmpeg.a (AIX) libdmbmpeg.sl (HP-UX) libdmbmpeg.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvFrameDataTo24BitRGB(
      unsigned char *RGB,
      DBvFrameData *fd,
      unsigned long dx,
      unsigned long dy
      );
```

Parameters

RGB (out)
The pointer to the target RGB buffer.

fd (in) The pointer to the frame data to be converted.

dx (in)
Frame width

dy (in)
Frame height

Error codes

MMDB_SUCCESS
API call processed successfully.

Examples

Convert a video frame from MPEG to 24-bit RGB:

```
#include <dmbshot.h>

rc = DBvFrameDataTo24BitRGB(RGB, &video->fd,
       video->dx, video->dy);
```

DBvGetError

Image	Audio	Video
		X

Returns a description of the last error. Call this API after any other API returns an error code.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvGetError(
      SQLINTEGER *sqlcode,
      char *errorMsgText
      );
```

Parameters

sqlcode (out)
The generic SQL error code.

errorMsgText (out)
The SQL error message text.

Error codes

MMDB_SUCCESS
API call processed successfully.

Examples

Get the last error, storing the SQL error code in `errCode` and the message text in `errMsg`:

```
#include <dmbvideo.h>

rc = DBvGetError(&errCode, &errMsg);
```

DBvGetFrame

Image	Audio	Video
		X

Gets the current frame in a video file. The frame data is returned in the DBvFrameData video structure.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbmpeg.lib	libdmbmpeg.a (AIX) libdmbmpeg.sl (HP-UX) libdmbmpeg.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvGetFrame(
      DBvIOType *video
      );
```

Parameters

video (update)
The pointer to the video structure.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_EOF
End of file reached.

Examples

Get the current frame in a video file:

```
#include <dmbshot.h>

rc = DBvGetFrame(video);
```

DBvGetInaccessibleFiles

Image	Audio	Video
		X

Returns the names of inaccessible files that are referenced in video columns of user tables. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SELECT privilege on enabled video columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvGetInaccessibleFiles(
      char *tableName,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

tableName (in)
A qualified, unqualified, or null table name. If a table name is specified, that table is searched for references to inaccessible files. If a null value is specified, all tables with the specified qualifier are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of inaccessible files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all inaccessible files referenced in video columns in the employee table:

```
#include <dmbvideo.h>
long idx;

rc = DBvGetInaccessibleFiles("employee",
       &count, &filelist);
```

DBvGetReferencedFiles

Image	Audio	Video
		X

Returns the names of files that are referenced in video columns of user tables. If a file is inaccessible (for example, its file name cannot be resolved using environment variable specifications), the file name is preceded with an asterisk. This API does not use the FILENAME field of the FILEREF data structure, and therefore sets it to NULL. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure.

Authorization

SELECT privilege on enabled video columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvGetReferencedFiles(
      char *tableName,
      long *count,
      FILEREF *(*fileList)
      );
```

Parameters

tableName (in)
A qualified, unqualified, or null table name. If a table name is specified, that table is searched for references to files. If a null value is specified, all tables owned by the current user ID are searched.

count (out)
The number of entries in the output list.

fileList (out)
A list of files that are referenced in the table.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List all files that are referenced in video columns in the employee table:

```
#include <dmbvideo.h>
long idx;

rc = DBvGetReferencedFiles("employee",
       &count, &filelist);
```

DBvInitShotControl

Image	Audio	Video
		X

Initializes the values in the shot control data structure.

Authorization

Insert, Control

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvInitShotControl(
      DBvShotControl *shotCtrl,
      );
```

Parameters

shotCtrl (in)
The pointer to the shot control data structure.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_ACCESS
Caller does not have proper access authority.

MMDB_RC_NOT_CONNECTED
The application does not have valid connection to a database.

Examples

Initialize the values in the shot control data structure:

```
#include <dmbshot.h>

rc = DBvInitShotControl(shotCtrl);
```

DBvInitStoryboardCtrl

Image	Audio	Video
		X

Initializes the values in the storyboard control data structure.

Authorization

Insert, Control

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvInitStoryboardCtrl(
      DBvStoryboardCtrl *sbCtrl,
      );
```

Parameters

shotCtrl (in)
The pointer to the shot control data structure.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_ACCESS
Caller does not have proper access authority.

MMDB_RC_NOT_CONNECTED
The application does not have valid connection to a database.

Examples

Initialize the values in the storyboard control data structure:

```
#include <dmbshot.h>

rc = DBvInitStoryboardCtrl(shotCtrl);
```

DBvInsertShot

Image	Audio	Video
		X

Inserts a shot into a shot catalog.

Authorization

Insert, Control

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvInsertShot(
      char *catalogName,
      DBvShotType *shot,
      DBvIOType *video,
      char *shotHandle,
      SQLHDBC hdbc
      );
```

Parameters

catalogName (in)
The name of the catalog.

shot (in)
The pointer to the extended shot to insert into the catalog.

shotHandle (in)
The shot handle.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_ACCESS
Caller does not have proper access authority.

MMDB_RC_NOT_CONNECTED
The application does not have valid connection to a database.

MMDB_RC_INVALID_CATALOG
The catalog is not valid or does not exist.

Examples

Insert a shot into a shot catalog called hotshots:

```
rc = DBvInsertShot(
"hotshots",      /* shot catalog name */
shot,                          /* pointer to shot structure */
video,                         /* pointer to video structure */
shotHandle,                    /* pointer to shot handle */
hdbc);                         /* database connection handle */
```

DBvIsColumnEnabled

Image	Audio	Video
		X

Determines whether a column has been enabled for video (DB2Video data). The application must be connected to a database before calling this API.

Authorization

SYSADM, DBADM, table owner, or SELECT privilege on the user table

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvIsColumnEnabled(
      char *tableName,
      char *colName,
      short *status
      );
```

Parameters

tableName (in)
A qualified or unqualified table name.

colName (in)
The name of a column.

status (out)
Indicates whether the column is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

0 MMDB_IS_NOT_ENABLED

-1 MMDB_INVALID_DATATYPE

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the video column in the employee table is enabled for video:

```
#include <dmbvideo.h>

rc = DBvIsColumnEnabled("employee",
       "video", &status);
```

DBvIsDatabaseEnabled

Image	Audio	Video
		X

Determines whether a database has been enabled for video (DB2Video data). The application must be connected to a database before calling this API.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvIsDatabaseEnabled(
      short *status
      );
```

Parameters

status (out)
Indicates whether the database is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

0 MMDB_IS_NOT_ENABLED

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the personnl database is enabled for video:

```
#include <dmbvideo.h>

rc = DBvIsDatabaseEnabled(&status);
```

DBvIsFileReferenced

Image	Audio	Video
		X

Returns a list of table entries in video columns that reference a specified file. The application must be connected to a database before calling this API.

It is important that you free up the resources allocated by this API after calling it. Specifically, you must free up the filelist data structure as well as the filename field in each entry in the filelist.

Authorization

SELECT privilege on enabled video columns in all searched user tables and associated administrative support tables

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvIsFileReferenced(
      char *tableName,
      char *fileName,
      long *count,
      FILEREF *(*tableList)
      );
```

Parameters

tableName (in)
A qualified, unqualified , or null table name. If a table name is specified, that table is searched for references to the specified file. If a null value is specified, all tables owned by the current user ID are searched.

fileName (in)
The name of the referenced file.

count (out)
The number of entries in the output list.

tableList (out)
A list of table entries that reference the specified file.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_MALLOC
System cannot allocate memory to return the results.

Examples

List the entries in video columns of the employee table that reference file /videos/ajones.mpg:

```
#include <dmbvideo.h>
long idx;

rc = DBvIsFileReferenced(NULL,
       "/videos/ajones.mpg",
       &count, &tableList);
```

DBvIsIndex

Image	Audio	Video
		X

Checks for the existence of the video index. The application must be connected to a database before calling this API.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbmpeg.lib	libdmbmpeg.a (AIX) libdmbmpeg.sl (HP-UX) libdmbmpeg.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvIsIndex(
      char *fileName,
      short *status
      );
```

Parameters

fileName (in)
the name of the referenced file.

status (out)
Indicates whether the index exists. A value of 1 means the index exists; a value of 0 means the index does not exist.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_ERROR
The status is not valid.

Examples

Check the existence of an index for the video file \videos\ajones.mpg:

DBvIsIndex

```
#include <dmbshot.h>

rc = DBvIsIndex("\videos\ajones.mpg", &status);
```

DBvIsTableEnabled

Image	Audio	Video
		X

Determines whether a table has been enabled for video (DB2Video data). The application must be connected to a database before calling this API.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvIsTableEnabled(
      char *tableName,
      short *status
      );
```

Parameters

tableName (in)
A table name.

status (out)
Indicates whether the table is enabled. This parameter returns a numerical value. The extender also returns a constant that indicates the status. The values and constants are:

1 MMDB_IS_ENABLED

0 MMDB_IS_NOT_ENABLED

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Determine if the employee table is enabled for video:

```
#include <dmbvideo.h>

rc = DBvIsTableEnabled("employee",
       &status);
```

DBvMergeShots

Image	Audio	Video
		X

Merges two shots into one shot. The resulting shot uses the shot handle and the starting frame of the first shot. The larger end frame of the two shots is used in the resulting shot. The row that the second shot handle points at is deleted.

Authorization

Control, Select, Delete, Update

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvMergeShots(
      char *catalogName,
      char *shotHandle1,
      char *shotHandle2,
      SQLHDBC hdbc
      );
```

Parameters

catalogName (in)
The name of the shot catalog.

shotHandle1 (in)
The handle of the first shot.

shotHandle2 (in)
The handle of the second shot.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
The application does not have valid connection to a database.

MMDB_RC_CANNOT_MERGE
Cannot merge shots.

MMDB_RC_INVALID_CATALOG
The catalog is not valid or does not exist.

Examples

Merge the shots with handles shotHandle1 and shotHandle2 in the hotshots catalog:

```
#include <dmbshot.h>

rc = DBvMergeShots("hotshots", shotHandle1,
       shotHandle2, hdbc);
```

DBvOpenFile

Image	Audio	Video
		X

Allocates space for a DBvIOType structure and opens the video file for pixel access. When the video is successfully opened, it points at the first frame number (frame 0).

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbmpeg.lib	libdmbmpeg.a (AIX) libdmbmpeg.sl (HP-UX) libdmbmpeg.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvOpenFile(
      DBvIOType **video,
      char *fileName,
      );
```

Parameters

video (out)
The pointer to the video structure pointer.

fileName (in)
The name of the video file to open.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_CANNOT_OPEN
Cannot open video file.

MMDB_RC_NO_MEMORY
Not enough memory.

MMDB_RC_NO_INDEX
No video random access index.

Examples

Open the video file \videos\ajones.mpg:

```
#include <dmbshot.h>

rc = DBvOpenFile(&videoa,
       "\videos\ajones.mpg");
```

DBvOpenHandle

Image	Audio	Video
		X

Allocates space for a DBvIOType structure and opens the video handle for pixel access. The structure points to the first frame number (frame 0). The video can be a BLOB. The video is copied to the temporary file in a directory specified by the DB2VIDEOTEMP environment variable. The isIdx flag is set based on the existence of the random access index.

Authorization

Select

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvOpenHandle(
      DBvIOType **video,
      DB2Video *videoHandle
      SQLHDBC hdbc
      );
```

Parameters

video (out)
The pointer to the video structure.

videoHandle (in)
The video handle.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_CANNOT_OPEN
Cannot open video file.

MMDB_RC_NO_MEMORY
Not enough memory.

MMDB_RC_NO_INDEX
No video random access index.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

MMDB_RC_INVALID_HANDLE
The video handle is not valid.

Examples

Open the videoHandle using the videoa pointer:

```
#include <dmbshot.h>

rc = DBvOpenHandle(&oa, videoHandle, hdbc);
```

DBvPlay

Image	Audio	Video
		X

Opens the video player on the client and plays a video. The video can be stored in a video column or an external file:

- If the video is stored in an external file, you can pass either the name of the file or the video handle to this API. The API uses the client environment variable DB2VIDEOPATH to resolve the file location. The file must be accessible from the client workstation.
- If the video is stored in a column, you must pass the video handle to the API. The application must be connected to the database and have read access to the table in which the video is stored.

If the video is stored in a column, the extender creates a temporary file and copies the content of the object from the column to the file. The extender might also create a temporary file if the video is stored in an external file and its relative filename cannot be resolved using the values in environment variables, or if the file is not accessible on the client machine. The temporary file is created in the directory specified in the DB2VIDEOTEMP environment variable. The extender then plays the video from the temporary file.

Authorization

Select authority on the user table, if playing a video from a column.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

Play a video stored in a column

```
long DBvPlay(
      char *playerName,
      MMDB_PLAY_HANDLE,
      DB2Video *videoHandle,
      waitFlag
      );
```

DBvPlay

Syntax

Play a video stored as a file

```
long DBvPlay(
      char *playerName,
      MMDB_PLAY_FILE,
      char *fileName,
      waitFlag
      );
```

Parameters

playerName (in)
The name of the video player. If set to NULL, the default video player specified by the DB2VIDEOPLAYER environment variable is used.

MMDB_PLAY_HANDLE (in)
A constant that indicates the video is stored in a column.

MMDB_PLAY_FILE (in)
A constant that indicates the video is stored as a file that is accessible from the client.

videoHandle (in)
The handle of the video. This parameter must be passed when you play a video in a column. If the video handle represents an external file, the client environment variable DB2VIDEOPATH is used to resolve the file location.

fileName (in)
The name of the file that contains the video. The API uses the client environment variable DB2VIDEOPATH to resolve the file location. The file must be accessible from the client workstation.

waitFlag (in)
A constant that indicates whether your program waits for the user to close the player before continuing. MMDB_PLAY_WAIT runs the player in the same thread as your application. MMDB_PLAY_NO_WAIT runs the player in a separate thread.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Play the video identified by the videoHandle. Run the default player in the same thread as the application:

```
#include <dmbvideo.h>

rc = DBvPlay(NULL, MMDB_PLAY_HANDLE, videoHandle,
       MMDB_PLAY_WAIT);
```

DBvPrepareAttrs

Image	Audio	Video
		X

Prepares user-supplied video attributes. This API is used when a video object with user-supplied attributes is stored or updated. The UDF code that runs on the server always expects data in "big endian"format, a format used by most UNIX platforms. If a video object is stored or updated in "little endian" format, that is, from a non-UNIX client, the DBvPrepare API must be used before the store or update request is made.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
void DBvPrepareAttrs(
      MMDBVideoAttrs *vidAttr
      );
```

Parameters

vidAttr (in)
The user-supplied attributes of the video.

Examples

Prepare user-supplied video attributes:

```
#include <dmbvideo.h>

DBvPrepareAttrs(&vidattr);
```

DBvReorgMetadata

Image	Audio	Video
		X

"Cleans up" video-related metadata tables, for example:

- Reclaims space that is no longer used in video metadata tables
- Deletes references in video metadata tables to video files that no longer exist

The application must be connected to a database before calling this API.

Authorization

Alter, Control, SYSADM, SYSCTRL, SYSMAINT, DBADM

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbvideo.lib	libdmbvideo.a (AIX) libdmbvideo.sl (HP-UX) libdmbvideo.so (Solaris)

Include file

dmbvideo.h

Syntax

```
long DBvReorgMetadata(
      char *tableName,
      );
```

Parameters

tableName (in)
A qualified, unqualified, or null table name. If a table name is specified, clean up is performed for video metadata tables associated with the specified user table. If a null value is specified, metadata tables for video columns in all tables owned by the current user ID are cleaned up.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NO_AUTH
Caller does not have the proper access authority.

MMDB_RC_NOT_CONNECTED
Application does not have valid connection to a database.

Examples

Clean up the metadata tables for video columns in the employee table:

```
#include <dmbvideo.h>

rc = DBvReorgMetadata("employee");
```

DBvSetFrameNumber

Image	Audio	Video
		X

Sets the current frame to a specified frame number.

Authorization

None

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbmpeg.lib	libdmbmpeg.a (AIX) libdmbmpeg.sl (HP-UX) libdmbmpeg.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvSetFrameNumber(
      DBvIOType *video
      unsigned long frameNumber
      );
```

Parameters

video (in)
The pointer to the video structure.

frameNumber (in)
Number of the requested frame.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_FRAME_NOT_FOUND
The requested frame could not be found.

MMDB_NO_INDEX
The video index does not exist.

Examples

Set the current frame to frame number 85 in a video file:

DBvSetFrameNumber

```
#include <dmbshot.h>

rc = DBvSetFrameNumber(video, 85);
```

DBvSetShotComment

Image	Audio	Video
		X

Updates the read-only comment within the shot.

Authorization

Control, Update

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvSetShotComment(
      char *catalogName,
      char *shotHandle,
      char *comment,
      SQLHDBC hdbc
      );
```

Parameters

catalogName (in)
The name of the catalog.

shotHandle (in)
The handle of the shot to be updated.

comment (in)
The new comment for the shot.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
The application does not have valid connection to a database.

MMDB_RC_CANNOT_UPDATE
The API cannot update the shot.

MMDB_RC_INVALID_CATALOG
The catalog is not valid or does not exist.

Examples

Change the remark describing the shot with the shotHandle in the catalog hotshots:

```
#include <dmbshot.h>

rc = DBvSetShotComment("hotshot", shotHandle,
       "This is a hot shot.", hdbc);
```

DBvUpdateShot

Image	Audio	Video
		X

Replaces the attributes of a video shot in the catalog. All the attributes, except for the comment, are replaced by the attributes in the DBvShotType structure. If the remark pointer is NULL, the existing remark remains unchanged.

Authorization

Control, Update

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbshot.lib	libdmbshot.a (AIX) libdmbshot.sl (HP-UX) libdmbshot.so (Solaris)

Include file

dmbshot.h

Syntax

```
long DBvUpdateShot(
      char *catalogName,
      DBvShotType *shot,
      char *shotHandle,
      SQLHDBC hdbc
      );
```

Parameters

catalogName (in)
The name of the catalog.

shot (in)
The pointer to shot information structure containing attributes of the shot.

shotHandle (in)
The shot handle.

hdbc (in)
The database handle from SQLConnect.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RC_NOT_CONNECTED
The application does not have valid connection to a database.

MMDB_RC_CANNOT_UPDATE
The API cannot update the shot.

MMDB_RC_NO_SHOT
The shot does not exist.

MMDB_RC_INVALID_CATALOG
The catalog is not valid or does not exist.

Examples

Update the attributes of a shot in the hotshots catalog:

```
#include <dmbshot.h>

rc = DBvUpdateShot("hotshots", shot,
       shothandle, hdbc);
```

DMBRedistribute (EEE Only)

Image	Audio	Video
X		

Redistributes QBIC feature data when a node is added to or removed from a nodegroup, or when a new partition map is established for a nodegroup.

Authorization

The API must be run from the instance-owning id.

Library file

Windows	AIX and Solaris
dmbrd.lib	libdmbrd.a (AIX)
	libdmbrd.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
long DMBRedistribute (
       char  *pNodeGroupName,
       char  DataRedistOption    /* ""continue"" use CONTINUE parameter */
       );
                                 /* blank:start redistribution */
```

Parameters

pNodeGroupName (in)
The name of the nodegroup to redistribute.

Error codes

MMDB_SUCCESS
API call processed successfully.

MMDB_RD_NO_CONTINUE
Resubmit without CONTINUE parameter.

MMDB_RD_CONTINUE
Resubmit with CONTINUE parameter.

Examples

Redistribute QBIC extender data in the groupone nodegroup:

DMBRedistribute

```
#include <dmbrdst.h>

rc = DMBRedistribute(groupone,"continue");
```

QbAddFeature

Image	Audio	Video
X		

Adds a feature to the currently opened catalog. In the database, QbAddFeature creates the feature table for the specified feature. After adding images to the image column in your user table, use the QbCatalogColumn API, which adds an entry for each image to the feature table and analyzes the images.

Authorization

Alter

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbAddFeature(
      QbCatalogHandle *cHdl,
      char *featureName
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

featureName (in)
The name of the feature. The following features are supplied with the Image extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

QbAddFeature

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

qbicECCatalogReadOnly
The catalog is open for read only.

qbicECDupFeature
The feature is already in the catalog.

qbiECinvalidFeatureClass
The feature you specified is not a valid name format.

Examples

Add the QbColorFeatureClass feature to the catalog identified by the handle CatHdl:

```
#include <dmbqbapi.h>

rc = QbAddFeature(CatHdl,
       QbColorFeatureClass);
```

QbCatalogColumn

Image	Audio	Video
X		

Catalogs the images in the image column of your user table that have not been cataloged. The API adds an entry for each image to the feature table, and then analyzes the images. When the API analyzes the image, it creates image data and stores it in the image's entry in the feature table. The default parameters for the features are used. The catalog must be open.

Authorization

Insert

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbCatalogColumn(
      QbCatalogHandle *cHdl
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

qbicECImageNotFound
The image cannot be found or accessed.

qbicECCatalogRO
The catalog is read-only.

QbCatalogColumn

Examples

```
#include <dmbqbapi.h>

rc = QbCatalogColumn(CatHdl);
```

QbCatalogImage

Image	Audio	Video
X		

Catalogs an entire image. The API adds an entry for the image to the feature table, and then analyzes the image. When the API analyzes the image, it creates image data and stores it in the image's entry in the feature table. The image handle must be from the image column associated with the current QBIC catalog. The image is cataloged according to the currently defined features classes. The default parameters for the features in the catalog are used.

Authorization

Insert

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbCatalogImage(
      QbCatalogHandle *cHdl,
      char *imgHandle
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

imgHandle (in)
The handle of the image.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

qbicECImageNotFound
The image cannot be found or accessed.

qbicECCatalogRO
The catalog is read-only.

Examples

Catalog an image identified by the handle Img_hdl:

```
#include <dmbqbapi.h>

rc = QbCatalogColumn(CatHdl, Img_hdl);
```

QbCloseCatalog

Image	Audio	Video
X		

Closes the catalog. The API frees the opened catalog handle and the allocated resources.

Authorization

Connect

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbCloseCatalog(
      QbCatalogHandle *cHdl
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

Examples

Close the catalog identified by the handle CatHdl:

```
#include <dmbqbapi.h>

rc = QbCloseCatalog(CatHdl);
```

QbCreateCatalog

Image	Audio	Video
X		

Creates a catalog in the currently connected database for the specified image column. The column must be enabled for image data. The API creates a name for the catalog, which is used as the qualifier.

Authorization

Alter

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbCreateCatalog(
     char *tableName,
     char *columnName,
     sqlinteger autoCatalog,
     char *reserved
     );
```

Parameters

tableName (in)
The name of the table that contains an image column.

columnName (in)
The name of the image column for which you are creating a catalog.

autoCatalog (in)
Indicates whether images added to the image column will be automatically cataloged, that is, added to the feature tables and analyzed. Specify 1 to set auto-cataloging on or 0 to set it off. If you don't set auto-cataloging on, use the QbCatalogColumn or QbCatalogImage APIs to catalog images you add to the image column.

reserved (in)
Not currently used.

Error codes

qbicECSqlError
An SQL error occurred.

qbicECNotEnabled
The database, table, or column is not enabled for the DB2Image data type.

qbicECDupCatalog
The catalog already exists.

qbicECIvalidFeature
One or more of the features are not valid.

Examples

Create a catalog for the images in the picture column of the employee table. Set auto-cataloging on:

```
#include <dmbqbapi.h>

rc = QbCreateCatalog("employee",
        "picture", 1);
```

QbDeleteCatalog

Image	Audio	Video
X		

Deletes the specified catalog from the current database. The API also disables the associated image column so it no longer can contain image data.

Authorization

Alter

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbDeleteCatalog(
      char *tableName,
      char *columnName
      );
```

Parameters

tableName (in)
The name of the table that contains the image column.

columnName (in)
The name of the image column associated with the catalog.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

qbicECCatalogInUse
The catalog was being used by someone else.

qbicECCatalogRO
The catalog is read-only.

qbicECSystem
A system error occurred.

qbicECSqlError
An SQL error occurred.

Examples

Delete the QBIC catalog associated with the picture column in the employee table:

```
#include <dmbqbapi.h>

rc=QbDeleteCatalog("employee", "picture");
```

QbGetCatalogInfo

Image	Audio	Video
X		

Returns a QbCatalogInfo structure that contains the following information:

- The name of the user table and the image column the catalog belongs to.
- The number of features included in the catalog.
- Whether auto-cataloging is set on.

Authorization

Select

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbGetCatalogInfo(
      QbCatalogHandle *cHdl,
      QbCatalogInfo *catInfo
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

catInfo (out)
The catalog information structure.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

Examples

Get information about the catalog identified by the handle CatHdl and return it in a structure called catInfo:

```
#include <dmbqbapi.h>

rc = QbGetCatalogInfo(CatHdl, &catInfo);
```

QbListFeatures

Image	Audio	Video
X		

Returns a list of the active features currently included in a catalog. The list is returned to a buffer you allocate.

Authorization

Select

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbListFeatures(
      QbCatalogHandle *cHdl,
      sqlinteger bufSize,
      sqlinteger *count,
      char *featureNames
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

bufSize (in)
The size of your buffer. To estimate the needed buffer size, you can use the feature count returned by the QbGetCatalogInfo API, and multiply the count by the length of the longest feature name. Feature names stored in the buffer are separated by a blank character.

count (out)
The number of returned feature names.

featureNames (out)
An array of feature names in your buffer.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

qbicECTruncateData
The returned data was truncated because the return buffer was too small.

Examples

Get a list of the active features in the catalog identified by the handle CatHdl. Store the information in the featureNames array.

First, calculate bufSize, which is the buffer size you need for the list. Use the QbGetCatalogInfo API to return the number of features in the catInfo structure. Then multiply that number by the constant qbiMaxFeatureName, which is the size of the longest feature name:

```
#include <dmbqbapi.h>

rc = QbGetCatalogInfo(CatHdl, &catInfo);

bufSize =
   catInfo.featureCount*qbiMaxFeatureName;

rc = QbListFeatures(CatHdl, bufSize,
       count, featureNames);
```

QbOpenCatalog

Image	Audio	Video
X		

Opens the QBIC catalog for a specific image column. You can open the catalog in read mode or update mode. The API returns a handle for the opened catalog. You then use the handle in other APIs to manage and populate the catalog.

Make sure you close the catalog after you are finished with it.

Authorization

Connect

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbOpenCatalog(
      char *tableName,
      char *columnName,
      sqlinteger mode,
      QbCatalogHandle *cHdl
      );
```

Parameters

tableName (in)
The table name containing the image column.

columnName (in)
The name of the image column.

mode (in)
The mode in which you are opening the catalog. Valid values are qbiRead and qbiUpdate.

cHdl (out)
A pointer to the handle of the catalog.

Error codes

qbicECCatalogNotFound
The catalog was not found.

qbicECCatalogInUse
The catalog was being used by someone else.

qbicECOpenFailed
The catalog could not be opened.

qbicECNotEnabled
The catalog is not enabled.

qbicECNoCatalogFound
No catalog was found.

qbicECSqlError
An SQL error occurred.

qbicECSystem
A system error occurred.

Examples

Open the catalog for the picture column in the employee table in read mode:

```
#include <dmbqbapi.h>

rc=QbOpenCatalog("employee", "picture",
     qbiread, &CatHdl);
```

QbQueryAddFeature

Image	Audio	Video
X		

Adds the specified feature to a query object.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

For OS/2 and Windows: dmbqqry.lib; for AIX, HP-UX, and Solaris: libdmbqqry.a

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryAddFeature(
      QbQueryHandle qObj,
      char *featureName
      );
```

Parameters

qObj (in)
The handle of the query object.

featureName (in)
The name of the query feature to be added. The following features are supplied with the image extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Error codes

qbiECinvalidQueryHandle
The query objecthandle you specified does not reference a valid query object.

qbiECunknownFeatureClass
The feature you specified is not a recognized feature class name.

qbiECinvalidFeatureClass
The feature you specified is not a valid name format.

qbiECfeaturePresent
The feature you specified is already a member of the query object.

qbiECallocation
The system cannot allocate enough memory.

Examples

Add the QbColorFeatureClass feature to the query object identified by the qoHandle handle:

```
#include <dmbqbapi.h>

rc = QbQueryAddFeature(qoHandle,
        "QbColorFeatureClass");
```

QbQueryCreate

Image	Audio	Video
X		

Creates a query object and returns a handle. You can use the handle with other APIs to manipulate the query object.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryCreate(
      QbQueryHandle qObj
      );
```

Parameters

qObj (out)
A pointer to the query handle. If unsuccessful, this handle is set to 0.

Error codes

qbiECallocation
The system cannot allocate enough memory.

Examples

Create a query object and return the handle in qoHandle:

```
#include <dmbqbapi.h>

rc = QbQueryCreate(&qoHandle);
```

QbQueryDelete

Image	Audio	Video
X		

Deletes a query object. The API releases all the memory used by the query object and any added features.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryDelete(
      QbQueryHandle qObj
      );
```

Parameters

qObj (in)
The handle of the query object.

Error codes

qbiECinvalidQueryHandle
The queryobject handle you specified does not reference a valid query.

Examples

Delete the query object identified by the handle qoHandle:

```
#include <dmbqbapi.h>
rc = QbQueryDelete(qoHandle);
```

QbQueryGetFeatureCount

Image	Audio	Video
X		

Returns the number of features added to the query object. The number can be either 0 or 1. The following features are supplied with the Image Extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryGetFeatureCount(
      QbQueryHandle qObj,
      sqlinteger* count
      );
```

Parameters

qObj (in)
The handle of the query object.

count (out)
The pointer to the variable to set to the number of features present.

Error codes

qbiECinvalidQueryHandle
The query object handle you specified does not reference a valid query object.

Examples

Return the number of features for the query objectidentified by the handle qoHandle:

```
#include <dmbqbapi.h>

rc = QbQueryGetFeatureCount(qoHandle,
       &count);
```

QbQueryGetFeatureWeight

Image	Audio	Video
X		

Returns the weight of the specified feature in a query object.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryGetFeatureWeight(
      QbQueryHandle qObj,
      sqldouble* weight
      );
```

Parameters

qObj (in)
The handle of the query object.

weight (out)
The pointer to the variable to get the feature weight.

Error codes

qbiECinvalidQueryHandle
The query object handle that you specified does not reference a valid query object.

Examples

Get the weight for the texture feature in a query object identified by the handle qoHandle:

```
#include <dmbqbapi.h>

rc = QbQueryGetFeatureWeight(qoHandle, "QbTextureFeatureClass, &weight);
```

QbQueryListFeatures

Image	Audio	Video
X		

Returns a current list of features in the query object. The API returns the list to a buffer you allocate. The following features are supplied with the Image Extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryListFeatures(
      QbQueryHandle qObj,
      sqlinteger bufSize,
      sqlinteger* count,
      char *featureNames
      );
```

Parameters

qObj (in)
The handle of the query object.

bufSize (in)
The size of the featureNames buffer. Use the qbiMaxFeatureName constant as the buffer size. Query object features are identified by a character string name.

count (out)
The number of the returned feature names.

featureNames (out)
The pointer to the array of feature names for the query object. The array is stored in the buffer that you allocate.

Error codes

qbiECinvalidQueryHandle
The query handle that you specified does not reference a valid query.

Examples

Return the number of features in the query object identified by the handle qoHandle. Use the qbiMaxFeatureName constant to determine the size of the buffer you need. Return the feature name to the feats buffer and the number of features to the retCount variable:

```
#include <dmbqbapi.h>

bufSize = qbiMaxFeatureName;

rc = QbQueryListFeatures(qoHandle, bufSize,
        &retCount, feats);
```

QbQueryNameCreate

Image	Audio	Video
X		

Stores and names a query object so that you can use it in a UDF. You can provide the name and description of the query object.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryNameCreate(
      QbQueryHandle qObj,
      char *name,
      char *description
      );
```

Parameters

qObj (in)
The handle of the query object.

name (in)
The name of the query object. The name can be up to 18 characters.

description (in)
A brief description of the query object, up to 250 characters.

Error codes

qbiECinvalidQueryHandle
The query object handle that you specified does not reference a valid query .

Examples

Give a name and description to the query object created with the QbQueryCreate API:

```
#include <dmbqbapi.h>

rc = QbQueryNameCreate(qHandle,
       "fshavgcol",
       "average color query, 10/15/96");
```

QbQueryNameDelete

Image	Audio	Video
X		

Deletes a query object. The query object must have been named and stored using the QbQueryNameCreate API.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryNameDelete(
      char *name
      );
```

Parameters

name (in)
The name of the query object you are deleting.

Error codes

qbiECinvalidQueryHandle
The queryobject handle that you specified does not reference a valid query object.

Examples

Delete the query object named fshavgcol:

```
#include <dmbqbapi.h>

rc = QbQueryNameDelete("fshavgcol",);
```

QbQueryNameSearch

Image	Audio	Video
X		

Searches the QBIC catalog for images that match the search criteria contained in a query object. The query object is identified by its name. The results, which include the image handles and QBIC search scores, are stored in a result array in the client memory. The results are sorted according to their scores.

Authorization

Select

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryNameSearch(
      char *qName,
      char *tableName,
      char *columnName,
      sqlinteger maxReturns,
      QbQueryScope* scope,
      sqlinteger resultType,
      sqlinteger* count,
      QbResult* returns
      );
```

Parameters

qName (in)
The name of the query object.

tableName (in)
The name of the table containing the column of images you want to search.

columnName (in)
The name of the image column. The column must be enabled for image data.

maxReturns (in)
The maximum number of images you want returned.

scope (in) (Reserved)
Must be set to 0 (NULL)

resultType (in) (Reserved)
Must be set to qbiArray.

count (out)
The pointer to the number of images returned. If zero is returned, make sure the image column is cataloged for all the features in the query object.

returns (out)
The pointer to the array of QbResult structures that hold the returned results. Make sure you allocate the buffer large enough to hold all the results you expect.

Error codes

qbiECinvalidQueryHandle
The query objecthandle you specified does not reference a valid query obje.

Examples

Run the query FSHAVGCOL against the cataloged images in the picture column of the employee table. Make sure that no more than six images are returned:

```
#include <dmbqbapi.h>

rc = QbQueryNameSearch("fshavgcol",
       "employee", "picture",
       6, 0, qbiArray, &count, &returns);
```

QbQueryRemoveFeature

Image	Audio	Video
X		

Removes a query feature from the query object and deallocates any associated memory. The following features are supplied with the Image Extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQueryRemoveFeature(
      QbQueryHandle qObj,
      char *featureName
      );
```

Parameters

qObj (in)
The handle of the query object.

featureName (in)
The name of the feature to be removed.

Error codes

qbiECinvalidQueryHandle
The query object handle you specified does not reference a valid query object.

qbiECinvalidFeatureClass
The feature you specified is not a valid name format.

qbiECfeatureNotPresent
The feature you specified is not a member of the query object.

Examples

Remove the QbColorFeatureClass feature from the query object identified by the handle qoHandle:

```
#include <dmbqbapi.h>

rc = QbQueryRemoveFeature(qoHandle,
        "QbColorFeatureClass");
```

QbQuerySearch

Image	Audio	Video
X		

Searches the QBIC catalog for images that match the search criteria contained in a query object. The query object is identified by a query object handle. The results, which include the image handles and their QBIC search scores, are stored in a result array in the client memory. They are sorted according to their scores.

Authorization

Select

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQuerySearch(
      QbQueryHandle qObj,
      char *tableName,
      char *columnName,
      sqlinteger maxReturns,
      QbQueryScope* scope,
      sqlinteger resultType,
      sqlinteger* count,
      QbResult* returns
      );
```

Parameters

qObj (in)
The handle of the query object.

tableName (in)
The name of the table containing the column of images you want to search.

columnName (in)
The name of the image column. The column must be enabled for image data.

maxReturns (in)
The maximum number of images you want returned.

scope (in) (Reserved)
Must be set to 0 (NULL)

resultType (in) (Reserved)
Must be set to qbiArray.

count (out)
The pointer to the number of images returned. If zero is returned, make sure the image column is cataloged for all the features in the query object.

returns (out)
The pointer to the array of QbResult structures that hold the returned results. Make sure you allocate the buffer large enough to hold all the results you expect.

Error codes

qbiECinvalidQueryHandle
The query object handle you specified does not reference a valid query object.

Examples

Query the cataloged images in the picture column of the employee table. Make sure that no more than six images are returned:

```
#include <dmbqbapi.h>

rc = QbQuerySearch(qHandle, "employee",
       "picture", 6, 0, qbiArray,
       &count, &returns);
```

QbQuerySetFeatureData

Image	Audio	Video
X		

Sets the source of the image data for a feature in a query object. You can set the data source only after adding a feature to a query object. The data source can be an image in a database table, file, or workstation buffer.

The following features are supplied with the Image extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQuerySetFeatureData(
      QbQueryHandle qObj,
      char *featureName,
      QbImageSource* imgSource
      );
```

Parameters

qObj (in)
The handle of the query object.

featureName (in)
The name of the feature to be set.

imgSource (in)
The pointer to the image source structure. If you specify 0 (NULL) for

imgSource, it means that the information should not be changed in the feature. See “Using data source structures” on page 148 for more information.

Error codes

qbiECinvalidQueryHandle
The query object handle you specified does not reference a valid query object.

qbiECunknownFeatureClass
The feature you specified is not a recognized feature class name.

qbiECinvalidFeatureClass
The feature you specified is not a valid name format.

qbiECfeatureNotPresent
The feature you specified is not a member of the query object.

qbiECfileUnreadable
The image source file cannot be found or read.

Examples

Set the data source for a histogram feature as a file on the client workstation:

```
#include <dmbqbapi.h>

QbQueryHandle qoHandle;
QbImageSource imgSource;

imgSource.sourceType = qbiSource_ClientFile;
strcpy(featureName, "QbColorHistogramFeatureClass");
strcpy(imgSource.clientFile, "/tmp/image.gif");

rc = QbQuerySetFeatureData(qoHandle, featureName, &imgSource);
```

QbQuerySetFeatureWeight

Image	Audio	Video
X		

Sets the weight of the specified feature in a query object.

Authorization

None.

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQuerySetFeatureWeight(
      QbQueryHandle qObj,
      sqldouble* weight
      );
```

Parameters

qObj (in)
The handle of the query object.

weight (out)
The pointer to the variable to set to the feature weight.

Error codes

qbiECinvalidQueryHandle
The query object handle that you specified does not reference a valid query object.

Examples

Set the weight for the color feature in a query object identified by the handle qoHandle:

```
#include <dmbqbapi.h>

weight=2.0
rc = QbQuerySetFeatureWeight(qoHandle, "QbColorFeatureClass", &weight);
```

QbQueryStringSearch

Image	Audio	Video
X		

Searches the QBIC catalog for images that match the search criteria contained in a query string. The results, which include the image handles and their QBIC search scores, are stored in a result array in the client memory. They are sorted according to their scores.

Authorization

Select

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqqry.lib	libdmbqqry.a (AIX) libdmbqqry.sl (HP-UX) libdmbqqry.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbQuerySearch(
      char *queryString,
      char *tableName,
      char *columnName,
      sqlinteger maxReturns,
      QbQueryScope* scope,
      sqlinteger resultType,
      sqlinteger* count,
      QbResult* returns
      );
```

Parameters

queryString (in)
The query string.

tableName (in)
The name of the table containing the column of images you want to search.

columnName (in)
The name of the image column. The column must be enabled for image data.

maxReturns (in)
The maximum number of images you want returned.

scope (in) (Reserved)
Must be set to 0 (NULL)

resultType (in) (Reserved)
Must be set to qbiArray.

count (out)
The pointer to the number of images returned. If zero is returned, make sure the image column is cataloged for all the features in the query string.

returns (out)
The pointer to the array of QbResult structures that hold the returned results. Make sure you allocate the buffer large enough to hold all the results you expect.

Error codes

qbiECinvalidQueryString
The query string you specified is invalid.

Examples

Query the cataloged images in the picture column of the employee table. Make sure that no more than six images are returned:

```
#include <dmbqbapi.h>

rc = QbQueryStringSearch("QbColorFeatureClass color=<255, 0, 0>"
       "employee",
       "picture", 6, 0, qbiArray,
       &count, &returns);
```

QbReCatalogColumn

Image	Audio	Video
X		

Re-analyze all existing images in the opened QBIC catalog for a new feature. The default parameters of the features are used. Use this API when you add a new feature to a catalog that already has images.

Authorization

Update, Insert

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbReCatalogColumn (
      QbCatalogHandle *cHdl
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

qbicECImageNotFound
The image cannot be found or accessed.

qbicECCatalogRO
The catalog is read-only.

Examples

Catalog all the images that have been not been cataloged:

QbReCatalogColumn

```
#include <dmbqbapi.h>

rc = QbReCatalogColumn(CatHdl);
```

QbRemoveFeature

Image	Audio	Video
X		

Deletes the feature table of the specified feature from the opened catalog.

Authorization

Alter

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbRemoveFeature(
      QbCatalogHandle *cHdl,
      char *featureName
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

featureName (in)
The name of the feature.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

qbicECCatalogReadOnly
The catalog is open for read only.

qbicECFeatureNotFound
The feature is not in the catalog.

qbiECinvalidFeatureClass
The feature you specified is not a valid name format.

Examples

Remove the QbColorHistogramFeatureClass feature from the catalog identified by the handle CatHdl:

```
#include <dmbqbapi.h>

rc=QbRemoveFeature(CatHdl,
     "QbColorHistogramFeatureClass");
```

QbSetAutoCatalog

Image	Audio	Video
X		

Automatically catalogs images that are imported into an image column. The API adds an entry for each image to the feature table, and then analyzes the images. When the API analyzes the image, it creates image data and stores it in the image's entry in the feature table.

If you don't set auto-cataloging on, use the QbCatalogColumn or QbCatalogImage API to catalog images after you add them to the image column.

Authorization

Alter

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbSetAutoCatalog(
      QbCatalogHandle *cHdl
      sqlinteger autoCatalog
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

autoCatalog (in)
Indicates whether images added to the image column will be automatically added to the feature tables and analyzed. Specify 1 to set auto-cataloging on or 0 to set it off.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

Examples

Set auto-cataloging on for the catalog identified by the handle CatHdl:

```
#include <dmbqbapi.h>

rc=QbSetAutoCatalog(CatHdl, 1);
```

QbUncatalogImage

Image	Audio	Video
X		

Removes an image from a catalog. The image handle must be from the image column that is associated with the opened QBIC catalog. The image will be removed from the opened catalog. The corresponding row in the image attribute table indicates the image is not cataloged.

Authorization

Delete

Library file

OS/2 and Windows	AIX, HP-UX, and Solaris
dmbqbapi.lib	libdmbqbapi.a (AIX) libdmbqbapi.sl (HP-UX) libdmbqbapi.so (Solaris)

Include file

dmbqbapi.h

Syntax

```
sqlinteger QbUncatalogImage(
      QbCatalogHandle *cHdl,
      char *imgHandle
      );
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

imgHandle (in)
The handle of the image. You can retrieve this handle from the user table.

Error codes

qbicECIvalidHandle
The catalog handle is not valid.

qbicECImageNotFound
The image cannot be found or accessed.

qbicECCatalogRO
The catalog is read-only.

Examples

Remove the image identified by the handle Img_hdl from the catalog identified by the handle CatHdl:

```
#include <dmbqbapi.h>

rc=QbUncatalogImage(CatHdl, Img_hdl);
```

Chapter 17. Administration Commands for the Client

This chapter describes how to enter DB2 extender administration commands for the client. It also gives reference information about each DB2 extender administration command for the client.

Entering DB2 extender administration commands

You can submit DB2 extender administration commands to the db2ext command-line processor in interactive mode or in command mode. Interactive mode is characterized by the db2ext prompt. In this mode, you can enter only DB2 extender administration commands. In command mode, you enter commands from the operating system command prompt; you can enter DB2 extender commands as well as DB2 commands and operating system commands.

Do not enter DB2 extender commands from the DB2 command prompt.

To start the db2ext command-line processor in interactive mode, do the following:

Client	Action
OS/2	Double-click on the DB2EXT Command Line Processor icon in the DB2 Extenders folder, or enter the DB2EXT command from the OS/2 command prompt.
AIX, HP-UX, Solaris	Enter the DB2EXT command from the operating system command prompt.
Windows 3.1	Double-click on the DB2EXT Command Line Processor icon in the DB2 Extenders program group, or enter the DB2EXT command from the Windows prompt.
Windows 95, Windows 98,Windows NT	Double-click on the DB2EXT Command Line Processor icon in the DB2 Extenders folder, or enter the DB2EXT command from the DB2 command window.

To end interactive mode, enter the quit or terminate command. The quit command ends interactive mode but maintains the current connection to DB2. The terminate command ends interactive mode and drops the current connection to DB2.

To submit DB2 extender commands in command mode, enter them from the operating system command line. You must precede each DB2 extender command with db2ext, for example:

```
db2ext enable database for db2image using mydataspace, myindxspace, mylongspace
```

Getting online help for DB2 extender commands

To get online help for all the DB2 extender commands, enter:

```
db2ext ?
```

ADD QBIC FEATURE

Image	Audio	Video
X		

Creates a feature table for the specified feature in the current catalog. Existing images in the catalog are not automatically reanalyzed by the Image Extender.

Authorization

Alter, Control, SYSADM, DBADM

Command syntax

Command parameters

feature_name
The name of the feature you are adding to the QBIC catalog. The following features are supplied with the Image extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Examples

Add the QbColorFeatureClass feature to the currently opened catalog:

```
add qbic feature qbcolorfeatureclass
```

Usage Notes

Connect to the database before using this command.

The catalog must be open.

CATALOG QBIC COLUMN

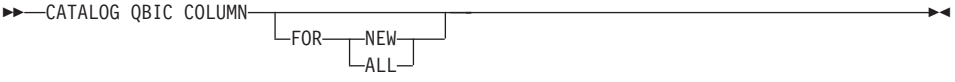
Image	Audio	Video
X		

Catalogs the images in the image column and updates the currently open QBIC catalog with feature data. You can update the catalog for all the images in the image column or for only the new images added to the image column since the last time the catalog was analyzed.

Authorization

Insert, Control, SYSADM, DBADM

Command syntax



Command parameters

None.

Examples

Catalog the new images in to the current catalog, that is, the images that have not been cataloged:

```
catalog qbic column new
```

Usage Notes

When NEW is specified, the QBIC extender updates the catalog only with the images that have not been cataloged. When ALL is specified, the QBIC extender analyzes every image in the image column for the current catalog. NEW is the default.

Connect to the database before using this command.

The catalog must be open.

CLOSE QBIC CATALOG

Image	Audio	Video
X		

Closes a QBIC catalog.

Authorization

None.

Command syntax

Command parameters

None.

Examples

Close the current catalog:

```
close qbic catalog
```

Usage Notes

The QBIC catalog must be open.

CONNECT

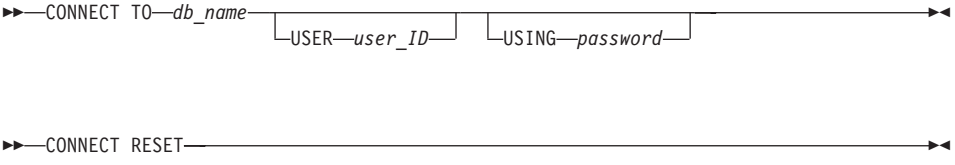
Image	Audio	Video
X	X	X

Connects to a database. The extenders require an independent connection to the database, separate from the DB2 connection.

Authorization

Connect

Command syntax



Command parameters

db_name
The name of the database.

user_ID
The user ID authorized to connect to the database.

password
The password for the user ID.

RESET
Disconnects the database after committing any pending changes.

Examples

Connect to the PERSONNL database. The user ID is anita and the password is anitapas:

```
connect to personnl user anita
   using anitapas
```

Usage Notes

The database is connected in SHARE mode.

Run this command before running any other extender commands.

CREATE QBIC CATALOG

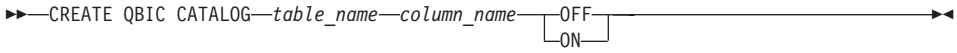
Image	Audio	Video
X		

Creates a QBIC catalog in the current database for the specified DB2IMAGE column. The extender automatically generates the catalog name.

Authorization

Alter, Control, SYSADM, DBADM

Command syntax



Command parameters

table_name
The name of the DB2IMAGE enabled table.

column_name
The name of the DB2IMAGE enabled column.

Examples

Create a QBIC catalog for the picture column in the employee table, with auto-cataloging set ON:

```
create qbic catalog employee picture on
```

Usage Notes

If you specify ON, the images imported into the column are automatically cataloged into the associated QBIC catalog. The default is OFF.

Connect to the database before using this command.

DELETE QBIC CATALOG

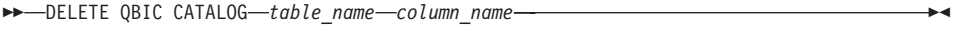
Image	Audio	Video
X		

Deletes a QBIC catalog, including all of the QBIC-search support data. The column associated with the catalog is disabled for any QBIC functions.

Authorization

Alter, Control, SYSADM, DBADM

Command syntax



Command parameters

table_name
The name of the DB2IMAGE enabled table.

column_name
The name of the DB2IMAGE enabled column.

Examples

Delete the catalog associated with the picture column in the employee table:

```
delete qbic catalog employee picture
```

Usage Notes

Connect to the database before using this command.

DISABLE COLUMN

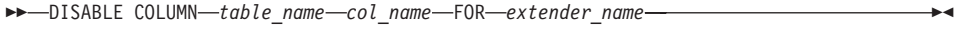
Image	Audio	Video
X	X	X

Disables the specified column from storing the specified media data.

Authorization

SYSADM, DBADM, Control, Alter

Command syntax



Command parameters

table_name
The name of the table in the current database.

col_name
The name of the column you want to disable.

extender_name
The name of the extender for which you want to disable the column. Valid extender names are db2image, db2audio, and db2video.

Examples

Disable the column photo in table employee so that it cannot hold image data:

```
disable column employee photo for db2image
```

Usage Notes

Connect to the database before using this command.

When you disable a column:

- The column can not store data for the specified extender. This does not affect whether other columns in the table are enabled or disabled for multimedia data types.
- The contents of the column entries are set to NULL and the corresponding rows in the administrative tables are deleted.
- The triggers associated with the column are dropped.

DISABLE DATABASE

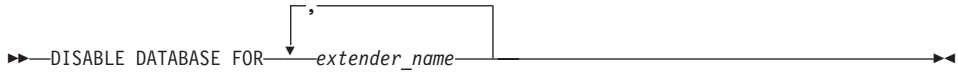
Image	Audio	Video
X	X	X

Disables the current database from storing media data.

Authorization

SYSADM, DBADM

Command syntax



Command parameters

extender_name
The name of the extender for which you want to disable the current database. Valid extender names are db2image, db2audio, and db2video.

Examples

Disable the current database from holding image data:

```
disable database for db2image
```

Usage Notes

Connect to the database before using this command.

When you disable a database, the system:

- Disables all the tables that are enabled for the specified extender only.
- Drops the UDFs administrative support tables for the specified extender.

DISABLE TABLE

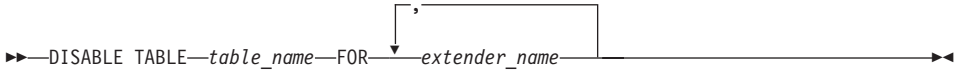
Image	Audio	Video
X	X	X

Disables the specified table from storing media data.

Authorization

SYSADM, DBADM, Control, Alter

Command syntax



Command parameters

table_name
The name of the table you want to disable in the current database.

extender_name
The name of the extender for which you want to disable the table. Valid extender names are db2image, db2audio, and db2video.

Examples

Disable the table employee from holding image data:

```
disable table employee for db2image
```

Usage Notes

Connect to the database before using this command.

When you disable a table, the system:

- Disables all the columns in the table that are enabled for the specified extender.
- Drops the administrative support tables associated with the table.

DISCONNECT SERVER AT NODENUM (EEE Only)

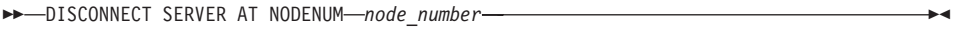
Image	Audio	Video
X	X	X

Disconnects the server from the specified node on all databases.

Authorization

SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax



Command parameters

node_number
The node you want to disconnect from the server.

Examples

Disconnect the server from all databases at node number 2:

```
disconnect server at nodenum 2
```

Usage Notes

To disconnect the server from all databases on all nodes, use the DMBSTOP command.

DISCONNECT SERVER FOR DATABASE (EEE Only)

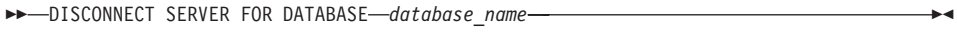
Image	Audio	Video
X	X	X

Disconnects the server from all nodes of the specified database.

Authorization

SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax



Command parameters

database_name
The database you want to disconnect from the server.

Examples

Disconnect the server from the database called MY_DATABASE:

```
disconnect server for database my_database
```

Usage Notes

To disconnect the server from all databases on all nodes, use the DMBSTOP command.

DISCONNECT SERVER FOR DATABASE AT NODENUM (EEE Only)

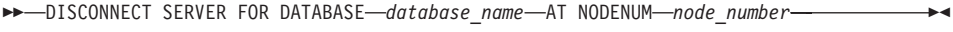
Image	Audio	Video
X	X	X

Disconnects the server from the specified database at the specified node.

Authorization

SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax



Command parameters

database_name
The database you want to disconnect from the server.

node_number
The node you want to disconnect from the server.

Examples

Disconnect the server from the database called MY_DATABASE at node number 2:

```
disconnect server for database my_database at nodenum 2
```

Usage Notes

To disconnect the server from all databases on all nodes, use the DMBSTOP command.

ENABLE COLUMN

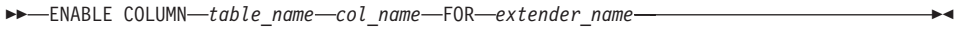
Image	Audio	Video
X	X	X

Enables the specified column to store media data.

Authorization

SYSADM, DBADM, Control, Alter

Command syntax



Command parameters

table_name
The name of the table in the current database.

col_name
The name of the column you want to enable.

extender_name
The name of the extender for which you want to enable the table. Valid extender names are db2image, db2audio, and db2video.

Examples

Enable the column photo in table employee to hold image data:

```
enable column employee photo for db2image
```

Usage Notes

Connect to the database before using this command.

ENABLE DATABASE

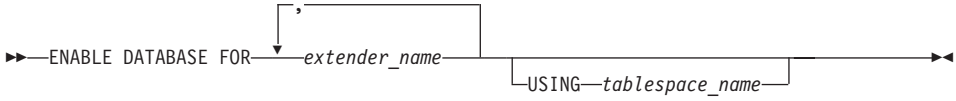
Image	Audio	Video
X	X	X

Enables the current database to store media data using the specified table space.

Authorization

SYSADM, SYSCTRL, DBADM

Command syntax



Command parameters

extender_name
The name of the extender for which you want to enable the current database. Valid extender names are db2image, db2audio, and db2video.

tablespace_name
The name of the table space, which is a collection of containers into which administrative tables are stored. The table space name has three parts as follows: *datats, indexts, longts*, where *datats* is the name of the table space in which metadata tables are created; *indexts* is the name of the table space in which indexes on the metadata tables are created; and *longts* is the name of the table space in which values of long columns in the metadata tables (such as those that contain LONG VARCHAR and LOB data types) are stored. If you provide a null value for any part of the table space name, the name of the default table space for that part is used. The table space specified should be defined on a nodegroup that includes all the nodes in the partitioned database system.

Examples

Enable the current database to hold image data:

```
enable database for db2image using mydataspace, myindxspace, mylongspace
```

Usage Notes

Connect to the database before using this command.

If the table space is not specified, the system uses the USERSPACE1 table space for the administrative tables.

ENABLE TABLE

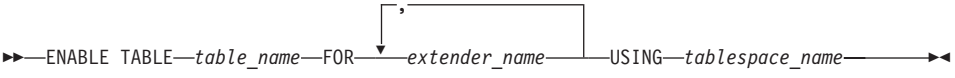
Image	Audio	Video
X	X	X

Enables the specified table to store media data using the specified table space.

Authorization

SYSADM, DBADM, Control, Alter

Command syntax



Command parameters

table_name
The name of the table in the current database you want to enable.

extender_name
The name of the extender for which you want to enable the table. Valid extender names are db2image, db2audio, and db2video.

tablespace_name
The name of the table space, which is a collection of containers into which administrative tables are stored. The table space specification has three parts as follows: *datats, indexts, longts*, where *datats* is the table space in which metadata tables are created; *indexts* is the table space in which indexes on the metadata tables are created; and *longts* is the table space in which values of long columns in the metadata tables (such as those that contain LONG VARCHAR and LOB data types) are stored. If you provide a null value for any part of the table space specification, the default table space for that part is used.

If you provide a null value for any part of the table space specification, the default table space for that part is used.

EEE Only: The tablespace specified should be in the same nodegroup as the user table.

Examples

Enable the table employee to hold image data:

```
enable table employee for db2image
   using mydataspace, myindxspace, mylongspace
```

Enable the table employee to hold image data. Use default table spaces:

```
enable table employee for db2image
```

Usage Notes

Connect to the database before using this command.

If the table space is not specified, the system uses the table space defined when the current database was enabled.

GET EXTENDER STATUS

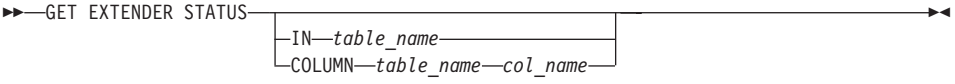
Image	Audio	Video
X	X	X

Displays the names of the extenders, if any, for which a column, table, or the current database is enabled.

Authorization

None

Command syntax



Command parameters

table_name
The name of the table in the current database.

col_name
The name of the column.

Examples

Display the names of the enabled extenders in the database:

```
get extender status
```

Display the status of the table employee:

```
get extender status in employee
```

Display the status of the column ADDRESS in the table employee:

```
get extender status column employee address
```

Usage Notes

Connect to the database before using this command.

GET INACCESSIBLE FILES

Image	Audio	Video
X	X	X

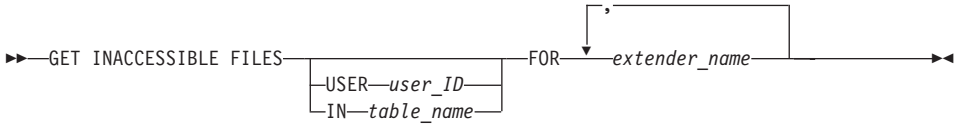
List all media files that are inaccessible and referenced by a table, tables with a specific qualifier, or all the tables in the current database.

Authorization

For all tables in the current database, that is, if you do not specify USER or IN: SYSADM, SYSCTRL, SYSMAINT, DBADM

For a particular table (if you specify IN) or tables that belong to a qualifier (if you specify USER): Select

Command syntax



Command parameters

user_ID
The qualifier of the tables in the current database whose inaccessible files you want to list.

table_name
The name of the table in the current database whose inaccessible files you want to list.

extender_name
The name of the extender. Valid extender names are db2image, db2audio, and db2video.

Examples

List all the image files referenced by tables in the database, but are inaccessible:

```
get inaccessible files
   for db2image
```

List all the image files referenced in tables with the qualifier anita, but are inaccessible:

```
get inaccessible files
   user anita for db2image
```

GET INACCESSIBLE FILES

List all the image files referenced by entries in the employee table, but are inaccessible:

```
get inaccessible files
   in employee FOR db2image
```

Usage Notes

Connect to the database before using this command.

If you specify a table the command lists inaccessible files for that table. If you specify a qualifier, the command lists inaccessible files for only those tables with that qualifier. If you specify neither, the command lists inaccessible files for all the tables in the current database.

GET QBIC CATALOG INFO

Image	Audio	Video
X		

Returns the following information about the currently opened catalog:

- The name of the user table and the image column with which the catalog is associated.
- The names of the features in the catalog.
- The number of features in the catalog.
- Whether auto-analyzing is on.

Authorization

Select, Control, SYSADM, DBADM

Command syntax

Command parameters

None.

Examples

Get information about the currently opened QBIC catalog:

```
get qbic catalog info
```

Usage Notes

Connect to the database before using this command.

The catalog must be open.

GET REFERENCED FILES

Image	Audio	Video
X	X	X

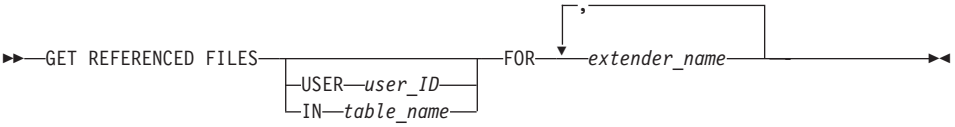
List all media files and the column names that reference them in a table, tables with a specific qualifier, or all the tables in the current database.

Authorization

For all tables in the current database, that is, if you do not specify USER or IN: SYSADM, SYSCTRL, SYSMAINT, DBADM

For a particular table (if you specify IN) or tables that belong to a qualifier (if you specify USER): Select

Command syntax



Command parameters

user_ID
The qualifier of the tables in the database whose referenced files you want to list. The command searches only tables with that qualifier.

table_name
The name of the table in the current database whose referenced files you want to list. The command searches that table only.

extender_name
The name of the extender. Valid extender names are db2image, db2audio, and db2video.

Examples

List all the image files referenced by table entries in all the tables in the database:

```
get referenced files
   for db2image
```

List all the image files referenced by entries in tables with the qualifier anita:

```
get referenced files
   user anita for db2image
```

List all the image files referenced by entries in the employee table:

```
get referenced files
   in employee for db2image
```

Usage Notes

Connect to the database before using this command.

If you do not specify any parameters, the command searches all the tables in the database.

GET SERVER STATUS

Image	Audio	Video
X	X	X

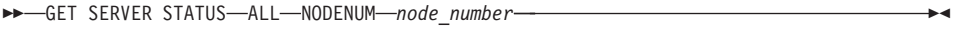
Displays the extender server status for the current database or for all databases.

EEE Only If a node is specified, the command displays the extender server status—for the current database or for all databases—at that node only.

Authorization

None

Command syntax



Command parameters

ALL Displays the status of all databases.

node_number
the number of the node. The command displays the status of this node. **(EEE Only)**

Examples

Display the status of the extender server for the current database:

```
get server status
```

Display the status of the extender server for all databases:

```
get server status all
```

Display the status of the extender server for node number 2 for all databases:

```
get server status all nodenum 2
```

Usage Notes

Connect to the database before using this command.

If you do not specify any parameters, the command displays the status of all nodes for the current database that are listed in the db2nodes.cfg file.

OPEN QBIC CATALOG

Image	Audio	Video
X		

Opens the catalog for the specified DB2IMAGE column. The database will always try to open the catalog with update mode. If the catalog is already in update mode, the catalog will be opened in read mode.

Authorization

Connect

Command syntax

Command parameters

table_name
The name of the DB2IMAGE enabled table.

column_name
The name of the DB2IMAGE enabled column.

Examples

Open the QBIC catalog for the picture column in the employee table:

```
open qbic catalog employee picture
```

Usage Notes

Connect to the database before using this command.

This command will cause any open catalog to close.

QUIT

Image	Audio	Video
X	X	X

Shuts down the db2ext command-line processor for command entry in interactive mode. The connection to DB2 is maintained, so you can still submit commands to the db2ext command-line processor in command mode.

Authorization

None

Command syntax

Command parameters

None.

Examples

Shut down the command-line interface for interactive mode:

```
quit
```

Usage Notes

QUIT maintains the connection to the database.

RECONNECT SERVER AT NODENUM (EEE Only)

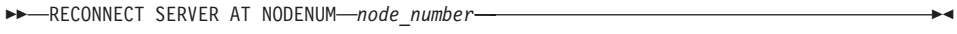
Image	Audio	Video
X	X	X

Reconnects the server to the specified node on all databases.

Authorization

SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax



Command parameters

node_number
The node you want to reconnect to the server.

Examples

Reconnect the server to all databases at node number 2:

```
reconnect server at nodenum 2
```

Usage Notes

To reconnect the server from all databases on all nodes, use the DMBSTART command.

RECONNECT SERVER FOR DATABASE (EEE Only)

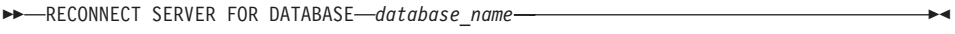
Image	Audio	Video
X	X	X

Reconnects the server to all nodes of the specified database.

Authorization

SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax



Command parameters

database_name
The database you want to reconnect to the server.

Examples

Reconnect the server to the database called MY_DATABASE:

```
disconnect server for database my_database
```

Usage Notes

To reconnect the server to all databases on all nodes, use the DMBSTART command.

RECONNECT SERVER FOR DATABASE AT NODENUM (EEE Only)

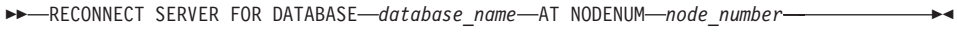
Image	Audio	Video
X	X	X

Reconnects the server to the specified database at the specified node.

Authorization

SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax



Command parameters

database_name
The database you want to reconnect to the server.

node_number
The node you want to reconnect to the server.

Examples

Reconnect the server to the database called MY_DATABASE at node number 2:

```
reconnect server for database my_database at nodenum 2
```

Usage Notes

To reconnect the server to all databases on all nodes, use the DMBSTART command.

REDISTRIBUTE NODEGROUP (EEE Only)

Image	Audio	Video
X		

Redistributes extender data when a node is added to or removed from a nodegroup, or when a new partition map is established for a nodegroup.

Authorization

SYSADM, DBADM

Command syntax

Command parameters

nodegroup
The name of the nodegroup you want to redistribute.

CONTINUE
If the redistribution process returns an error, you can re-run the command with or without the CONTINUE parameter according to the instructions provided by the command response. This option instructs the system to continue from where it stopped, rather than starting from the beginning.

Examples

Redistribute the nodegroup called my_nodegroup:

```
redistribute nodegroup my_nodegroup
```

Usage Notes

Connect to the database before using this command.

The CONTINUE parameter should not be used the first time you run the REDISTRIBUTE NODEGROUP command following DB2's REDISTRIBUTE command. If it is used the first time, then an error is logged and redistribution starts from the beginning.

To maintain data integrity, you must redistribute one nodegroup at a time. Wait until one nodegroup has finished redistribution before starting another.

If REDISTRIBUTE NODEGROUP fails, you can refer to the file "redist.log" for a detailed explanation in one of the following directories:

- **Unix:** /<home-instance>/dmb/redist
- **Windows NT:** \\<instance_owning_machine>\DB2<instance_name>\,instance_name>\dmb\redist

REMOVE QBIC FEATURE

Image	Audio	Video
X		

Deletes the feature table of the specified feature from the opened catalog.

Authorization

Alter, Control, SYSADM, DBADM

Command syntax

Command parameters

feature_name
The name of the feature you are removing from the QBIC catalog. The following features are supplied with the Image extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Examples

Remove the QbColorFeatureClass feature from the currently opened catalog:

```
remove qbic feature qbcolorfeatureclass
```

Usage Notes

Connect to the database before using this command.

The catalog must be open.

REORG

Image	Audio	Video
X	X	X

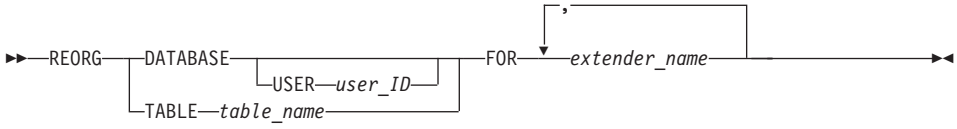
Clean up the administrative tables (administrative table and the attribute table) associated with a specific table, tables with a specific qualifier, or all tables in the current database.

Authorization

For a specific table (if you run REORG TABLE), or tables with a specific qualifier (if you run REORG DATABASE): SYSADM, SYSCTRL, SYSMAINT, DBADM, Control

For all tables in the database (if you run REORG DATABASE): SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax



Command parameters

user_ID
The qualifier of the tables.

table_name
The name of the table in the current database whose administrative tables you want to clean up.

extender_name
The name of the extender. Valid extender names are db2image, db2audio, and db2video.

Examples

Reorganize and clean up the image administrative tables for the current database:

```
reorg database for db2image
```

Reorganize and clean up the image administrative tables in all tables with the qualifier anita:

```
reorg database user anita for db2image
```

REORG

Reorganize and clean up the image administrative tables for the employee table:

```
reorg table employee for db2image
```

Usage Notes

Connect to the database before using this command.

SET QBIC AUTOCATALOG

Image	Audio	Video
X		

Automatically catalogs images when they are imported into a column. The images are added to the QBIC catalog associated with the column.

Authorization

Alter, Control, SYSADM, DBADM

Command syntax

Command parameters

None.

Examples

Set auto-cataloging on:

```
set qbic autocatalog on
```

Usage Notes

The QBIC catalog must be open.

START SERVER (Non-EEE Only)

Image	Audio	Video
X	X	X

Starts the extender server for the current database.

Authorization

SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax

Command parameters

None.

Examples

Start the extender server for the current database:

```
start server
```

Usage Notes

Connect to the database before using this command.

STOP SERVER (Non-EEE Only)

Image	Audio	Video
X	X	X

Stops the extender server for the current database.

Authorization

SYSADM, SYSCTRL, SYSMAINT, DBADM

Command syntax

Command parameters

None.

Examples

Stop the extender server for the current database:

```
stop server
```

Usage Notes

Connect to the database before using this command.

TERMINATE

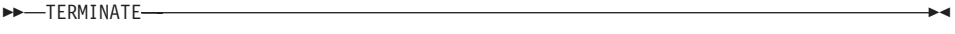
Image	Audio	Video
X	X	X

Shuts down the db2ext command-line processor and drops the connection to DB2.

Authorization

None

Command syntax



Command parameters

None.

Examples

Shut down the db2ext command-line processor:

```
quit
```

Usage Notes

TERMINATE drops the connection to the database.

Chapter 18. Administration Commands for the Server

The commands in this chapter are run on the command line of the server's operating system. Do not run them from the DB2 command line or the db2ext command line. Run the DMBSTART command whenever you shut down and restart your server system.

EEE Only: You can also issue the DMBSTART and DMBSTOP server commands in a multipartition database environment. When you issue a server command in a multipartition database environment, the command applies to all nodes, unless you include a node number, in which case the command applies only to the specified node.

EEE Only: The DMBSTAT command cannot be run in a multipartition environment. Server status can be checked in a multipartition environment by running the client command GET SERVER STATUS ALL.

DMBSTART

Image	Audio	Video
X	X	X

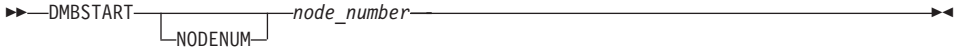
Starts all the extender services for the extender instance.

EEE Only: If a node is specified, the command starts extender services at that node only. DMBSTART also initiates the Node Create/Drop function at each node. The Node Create/Drop function checks to make sure that the nodes defined for DB2 match the nodes defined for the Extenders. If they do not match, the Node Create/Drop function adds or removes nodes as needed.

Authorization

SYSADM

Command syntax



Command parameters

node_number
The node at which you want to start extender services. **(EEE Only)**

Examples

Start extender services:

```
dmbstart
```

Start extender services at node number 2:

```
dmbstart nodenum 2
```

Usage Notes

Run this command:

- While logged on as instance owner (on AIX, HP-UX, or Solaris).
- From a window where the DB2INSTANCE environment variable is the same as the instance you want to start (on OS/2 or Windows NT).
- Whenever you shut down and restart your server system.

In a single-partition environment, DMBSTART also starts the DB2 instance if it is not running.

EEE Only: In a multipartition environment, DMBSTART does not start the DB2 instance. You must start DB2 before running DMBSTART in a partitioned environment.

DMBSTAT

Image	Audio	Video
X	X	X

Displays which databases are enabled and if the extender services for those databases are up and running.

Authorization

None

Command syntax

Command parameters

None.

Examples

Display the status of extender services:

```
dmbstat
```

DMBSTOP

Image	Audio	Video
X	X	X

Stops the extender services for the extender instance.

EEE Only: If a node is specified, DMBSTOP stops extender services only at that node.

Authorization

SYSADM

Command syntax

Command parameters

node_number
The node at which you want to stop the extender services. **(EEE Only)**

Examples

Stop the extender services:

```
dmbstop
```

Stop the extender services at node number 2:

```
dmbstop nodenum 2
```

Usage Notes

Run this command:

- While logged on as instance owner (on AIX, HP-UX, or Solaris).
- From a window where the DB2INSTANCE environment variable is the same as the instance you want to stop (on OS/2 or Windows NT).

DMBSTOP does not stop the DB2 instance.

EEE Only: Do not run DMBSTOP at a specific node except while operating in maintenance mode. In addition, you need to make sure that no extender activities will be triggered on this node while it is turned off. Otherwise, you may encounter unexpected behavior.

DMBSTOP

Chapter 19. Diagnostic Information

All embedded SQL statements in your program and DB2 CLI calls in your program, including those that invoke DB2 extender UDFs, generate codes that indicate whether the embedded SQL statement or DB2 CLI call executed successfully. Other DB2 extender APIs, such as administrative APIs, also return codes that indicate success or lack of success. Your program should check and respond to these return codes.

Your program can also retrieve information that supplements these codes. This includes SQLSTATE information and error messages. You can use this diagnostic information to isolate and fix problems in your program.

Occasionally the source of a problem cannot be easily diagnosed. In these cases, you might need to provide information to service personnel to isolate and fix the problem. The DB2 extenders include a trace facility that records extender activity. The trace information can be valuable input to service personnel. You should use the trace facility only under instruction from IBM service personnel.

This chapter describes how to access this diagnostic information. It describes:

- How to handle DB2 extender UDF return codes and API return codes.
- How to control tracing

It also lists and describes the SQLSTATEs and error messages that can be returned by the extenders.

Handling UDF return codes

Embedded SQL statements return codes in the SQLCODE, SQLWARN, and SQLSTATE fields of the SQLCA structure. This structure is defined in an SQLCA include file. (For more information about the SQLCA structure and SQLCA include file, see *DB2 Embedded SQL Programming Guide*.)

DB2 CLI calls return SQLCODE and SQLSTATE values that you can retrieve using the SQLError function. (For more information about retrieving error information with the SQLError function, see *CLI Guide and Reference*.)

An SQLCODE value of 0 means successful execution (with possible warning conditions). A positive SQLCODE value means that the statement was successfully executed but with a warning. (Embedded SQL statements return

the warning associated with 0 or positive SQLCODE values in the SQLWARN field.) A negative SQLCODE value means that an error condition occurred.

DB2 associates a message with each SQLCODE value. If a DB2 extender UDF encounters a warning or error condition, it passes associated information to DB2 for inclusion in the SQLCODE message.

SQLSTATE values contains codes that supplement the SQLCODE messages. See “SQLSTATE codes” on page 519 for a description of each SQLSTATE code returned by the DB2 Extenders.

Embedded SQL statements and DB2 CLI calls that invoke DB2 extender UDFs might return SQLCODE messages and SQLSTATE values that are unique to these UDFs, but DB2 returns these values in the same way as it does for other embedded SQL statements or other DB2 CLI calls. Thus, the way you access these values is the same as for embedded SQL statements or DB2 CLI calls that do not invoke DB2 extender UDFs.

See “SQLSTATE codes” on page 519 for the SQLSTATE values and the message number of associated messages that can be returned by the extenders. See “Messages” on page 522 for information about each message.

Handling API return codes

Each DB2 extender API call returns a code. A return code of 0 indicates that the API call was processed successfully. A return code other than 0, indicates that the API call was processed successfully but encountered a warning condition, or could not be processed successfully because of an error condition.

“Chapter 16. Application Programming Interfaces” on page 265 lists the symbolic value for and describes each code that can be returned by the DB2 extender APIs.

You can retrieve additional information about errors encountered by an API. Use the DB*x*GetError API to retrieve this additional information, where *x* is a for the Audio Extender, i for the Image Extender, and v for the Video Extender. The DB*x*GetError API returns the SQL error code and associated message for the last DB2 extender API that encountered an error. See *DB2 Messages ReferenceDB2 Messages and Codes* for information about SQL error codes. See “Messages” on page 522 for information about each message that can be returned by the DB*x*GetError API.

For example, the following statements in a C application program enable a table for the Audio Extender and then check for errors.

```
rc=DBaEnableTable((char *)NULL, "employee");

rc=DBaGetError(&errCode, &errMsg);
```

SQLSTATE codes

Table 16 lists and describes the SQLSTATE values that can be returned by the DB2 extenders. The description of each SQLSTATE value includes its symbolic representation. The table also lists the message number associated with each SQLSTATE value. See “Messages” on page 522 for information about each message.

Table 16. SQLSTATE codes and associated message numbers

SQLSTATE	Message No.	Description
00000		MMDB_SQLSTATE_OK Successful
01H01	DMB0211W	MMDB_SQLSTATE_WARN_NO_OVERWRITE The file overwrite does not take place
38A00	DMB0101E	MMDB_SQLSTATE_AUDIO_NULL_PARM Input parameter to the UDF cannot be null
38A02	DMB0209E	MMDB_SQLSTATE_AUDIO_OPEN_HDR_ERROR Error occurred while opening audio file header
38A03	DMB0209E	MMDB_SQLSTATE_AUDIO_BAD_WAVE_HDR Invalid wave file supplied
38V00	DMB0101E	MMDB_SQLSTATE_VIDEO_NULL_PARM Input parameter to the UDF cannot be null
38V02	DMB0051E	MMDB_SQLSTATE_VIDEO_OPEN_HDR_ERROR Error occurred while opening video file header
38V03	DMB0105E	MMDB_SQLSTATE_VIDEO_BAD_MPEG1_HDR Invalid mpeg1 file supplied
38V04	DMB0104E	MMDB_SQLSTATE_VIDEO_BLOB_TOO_SHORT Video buffer supplied is too small
38V05	DMB0106E	MMDB_SQLSTATE_VIDEO_BAD_AVI_HDR Invalid AVI file supplied
38V07	DMB0106E	MMDB_SQLSTATE_VIDEO_BAD_QT_HDR Invalid Quicktime file supplied
38600	DMB0075E DMB0101E DMB0102E DMB0103E DMB0210E	MMDB_SQLSTATE_INVALID_INPUT Input parameter to the UDF is not valid

Table 16. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38601	DMB0009E	MMDB_SQLSTATE_MALLOC_FAIL Memory allocation failed
38602	DMB0386E	MMDB_SQLSTATE_CANNOT_COLLOCATE Cannot collocate with user data
38603	DMB0077E	MMDB_SQLSTATE_READ_FILE_FAIL Cannot read from file
38604	DMB0080E	MMDB_SQLSTATE_WRITE_FILE_FAIL Cannot write to file
38610	DMB0070E	MMDB_SQLSTATE_INVALID_HANDLE Media column contains invalid data
38611	DMB0073E	MMDB_SQLSTATE_INVALID_SESSION_HANDLE Invalid UDF session handle
38612	DMB0074E	MMDB_SQLSTATE_INVALID_STATEMENT_HANDLE Invalid UDF statement handle
38613	DMB0083E	MMDB_SQLSTATE_INVALID_IMPORT_REQUEST The request for import is not valid
38615	DMB0071E	MMDB_SQLSTATE_CONNECT_FAIL Error occurred while connecting to database
38617	DMB0071E	MMDB_SQLSTATE_ALLOC_STMT_FAIL Error occurred while allocating a new statement handle
38618	DMB0208E DMB0138E	MMDB_SQLSTATE_FREE_STMT_FAIL Error occurred while freeing statement
38619	DMB0208E DMB0132E	MMDB_SQLSTATE_BIND_FAIL Error occurred while binding
38620	DMB0208E	MMDB_SQLSTATE_BIND_COLUMN_FAIL Error occurred while binding a column
38621	DMB0208E	MMDB_SQLSTATE_BIND_FILE_FAIL Error occurred while binding file
38622	DMB0208E DMB0132E	MMDB_SQLSTATE_SET_PARAM_FAIL Error occurred while setting parameter
38623	DMB0208E DMB0131E	MMDB_SQLSTATE_PREPARE_FAIL Error occurred while preparing an SQL statement
38624	DMB0208E DMB0133E	MMDB_SQLSTATE_EXECUTE_FAIL Error occurred while executing a statement
38625	DMB0208E DMB0133E	MMDB_SQLSTATE_EXEC_DIRECT_FAIL Error occurred while directly executing SQL statement in UDF
38626	DMB0208E DMB0133E	MMDB_SQLSTATE_FETCH_FAIL Error occurred while retrieving next row of data

Table 16. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38627	DMB0208E	MMDB_SQLSTATE_COMMIT_FAIL Error occurred while committing the transaction
38628	DMB0208E	MMDB_SQLSTATE_GET_LENGTH_FAIL Error occurred while retrieving the length of a string value
38629	DMB0208E	MMDB_SQLSTATE_GET_SUBSTRING_FAIL Error occurred while retrieving a portion of a string value
38650	DMB0077E DMB0079E	MMDB_SQLSTATE_COPY_BLOB_2_FILE_FAIL Error occurred while copying BLOB to a file
38651	DMB0086E	MMDB_SQLSTATE_BLOB_BUFFER_TOO_SMALL The buffer supplied is too small
38652	DMB0082E	MMDB_SQLSTATE_BUILD_HANDLE Error occurred while constructing media column data
38653	DMB0083E	MMDB_SQLSTATE_INVALID_INSERT_VIA_SELECT The request for insert via select is not valid
38654	DMB0081E	MMDB_SQLSTATE_INVALID_OFFSET_SIZE The offset size is not valid
38655	DMB0068E	MMDB_SQLSTATE_METATABLE_DOESNOT_EXIST The required metadata table does not exist
38670	DMB0134E DMB0103E	MMDB_SQLSTATE_UNKNOWN_FORMAT The stored media has unknown format
38671	DMB0135E	MMDB_SQLSTATE_CREATE_THUMBNAIL_FAIL Error occurred while creating the thumbnail
38672	DMB0114E	MMDB_SQLSTATE_FORMAT_CONVERSION_FAIL Error occurred while converting the file format
38673	DMB0363E	MMDB_SQLSTATE_INVALID_UPDATE Error occured when an update UDF was invoked without referencing a table
38674	DMB0361E	MMDB_SQLSTATE_NOT_ENABLED Error occured when an import UDF was applied to a column which was not enabled for the extender
38675	DMB0129E	MMDB_SQLSTATE_VIDEO_INTERNAL Internal error in Video Extender UDFs
38676	DMB0129E	MMDB_SQLSTATE_AUDIO_INTERNAL Internal error in Audio Extender UDFs
38677	DMB0129E	MMDB_SQLSTATE_IMAGE_INTERNAL
38678	DMB0089E DMB0208E	MMDB_SQLSTATE_BASE_INTERNAL_ERROR Internal error in common layer

Table 16. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38681	DMB0108E	MMDB_SQLSTATE_IMPORT_ENV_NOT_SETUP Environment variable for import is not set up correctly
38682	DMB0111E	MMDB_SQLSTATE_STORE_ENV_NOT_SETUP Environment variable for store operation is not set up correctly
38683	DMB0107E	MMDB_SQLSTATE_EXPORT_ENV_NOT_SETUP Environment variable for export operation is not set up correctly
38684	DMB0117E	MMDB_SQLSTATE_TEMP_ENV_NOT_SETUP Environment variable for creating temporary files is not set up correctly
38686	DMB0109E	MMDB_SQLSTATE_CANT_RESOLVE_IMPORT_FILE Error occurred while resolving import file name
38687	DMB0112E	MMDB_SQLSTATE_CANT_RESOLVE_STORE_FILE Error occurred while resolving store file name
38688	DMB0110E	MMDB_SQLSTATE_CANT_RESOLVE_EXPORT_FILE Error occurred while resolving export file name
38689	DMB0116E	MMDB_SQLSTATE_CANT_CREATE_TMP_FILE Error occurred while creating temporary file
38690	DMB0076E	MMDB_SQLSTATE_OPEN_IMPORT_FILE_FAIL Cannot open import file
38691	DMB0115E	MMDB_SQLSTATE_OPEN_STORE_FILE_FAIL Cannot open import file
38692	DMB0114E	MMDB_SQLSTATE_OPEN_EXPORT_FILE_FAIL Cannot open export file
38693	DMB0118E	MMDB_SQLSTATE_OPEN_TEMP_FILE_FAIL Cannot open temporary file
38694	DMB0117E	MMDB_SQLSTATE_OPEN_CONTENT_FILE_FAIL Cannot open content file
38695	DMB0135E	MMDB_SQLSTATE_OPEN_THUMBNAIL_FILE_FAIL Cannot open thumbnail file
38696	DMB0135E	MMDB_SQLSTATE_READ_THUMBNAIL_FILE_FAIL Cannot read thumbnail file
38697	DMB0207E	MMDB_SQLSTATE_OVERWRITE_NOT_ALLOWED The overwrite operation could not be performed

Messages

DMB0001E The DB2 Extenders server was not connected. Reason: ″<code>″

Cause: The attempted operation attempted requires the DB2 extenders services to be running.

Action: On the server, run the DMBSTART command on the command line for the operating system.

DMB0003W The DB2 extenders trace facility is running for this session.

Cause: The trace facility uses up system resources.

Action: If the performance of your system is affected, you might want to turn off tracing.

DMB0004I This program can be run only by the instance owner: ″<name>″.

Cause: The DB2 extender servers must be started from the user ID under which the instance was created.

Action: Run the DMBSTART command from the user ID under which the instance was created.

DMB0005E The current database is not enabled for the ″<extender-name>″ extender.

Cause: An operation was attempted that requires the database to be enabled for a specific DB2 extender. For example, if you want to enable a table for DB2IMAGE data, you must first enable the database in which the table is stored for DB2IMAGE data.

Action: Enable the database for the extender data type you want and try again.

DMB0006E User ″<name>″ is not authorized to call this API.

Cause: The call to an application programming interface was attempted from a user ID that does not have the level of authority required for that API.

Action: Either run the application from another user ID, or get the database administrator to change the level of authority for the initial user ID.

DMB0007E User table ″<table-name>″ is not enabled for extender ″<extender-name>″.

Cause: The table on which the operation was attempted is not enabled for that DB2 extender. For example, a table must be enabled to hold audio data before a column in the table can be enabled for audio.

Action: Make sure the table is enabled for the extender first. Then enable the column.

DMB0008E An error occurred while running the stored procedure ″<name>″.

Cause: Either there is an error in the stored procedure identified in the message, or there is a problem with the environment.

Action: Verify your application and try again.

DMB0009E Memory allocation error.

Cause: The system was unable to allocate memory required to support the attempted operation.

Action: Verify that your system has enough memory to complete the operation.

DMB0010E The ″<extender-name>″ extender has been previously defined for the UDT ″<name>″.

Cause: The name of the user-defined type (UDT) has already been used for a UDT that was defined for another DB2 extender.

Action: Choose another name for your UDT.

DMB0011E User column ″<column-name>″ cannot be enabled for the ″<extender-name>″ extender. The definition of the user column is not compatible with the distinct type ″MMDBSYS.<name>″ associated with the extender.

Cause: The indicated column is not defined for the data type shown in the message, so it cannot be enabled for that extender.

Action: Make sure the column you want to enable has been defined using the data type that corresponds to the extender.

DMB0012E The specified user table ″<table-name>″ does not exist.

Cause: No table exists with the specified name.

Action: Check the name of the table and whether the table exists.

DMB0013E Column ″<column-name>″ is not defined for table ″<table-name>″.

Cause: The attempted operation referenced a column name that does not exist in the identified table.

Action: Check the names of the table and the column.

DMB0014W Column ″<column-name>″ in user table ″<table-name>″ is already enabled for the ″<extender-name>″ extender.

Cause: An attempt was made to enable the column for an extender for which the column is already enabled.

Action: No action is required.

DMB0015W The database is already enabled for extender ″<extender-name>″.

Cause: An attempt was made to enable the database for an extender for which the database is already enabled.

Action: No action is required.

DMB0016W User table ″<table-name>″ is already enabled for the ″<extender-name>″ extender.

Cause: An attempt was made to enable a table for an extender for which the table is already enabled.

Action: No action is required.

DMB0017E User table ″<table-name>″ is already enabled for the ″<extender-name>″ extender. But at least one of the associated metadata tables ″<table-name>″ or ″<table-name>″ doesn't exist.

Cause: One or more of the administrative support (metadata) tables associated with the table has been damaged or destroyed. Without these metadata tables, the user table cannot be used for data of that extender's type.

Action: Disable the user table and re-enable it for the extender.

DMB0018E The system cannot create a unique trigger name for column ″<column-name>″ in table ″<table-name>″.

Cause: When the system tried to enable the column in the user table, an error occurred during the creation of triggers used by the DB2 extenders.

Action: Repeat the operation. If the error occurs again, contact your database administrator, then contact IBM service.

DMB0019I ″<Count>″ files are referenced in table ″<table-name>″ for extender ″<extender-name>″.

Cause: The message displays the number of external media files that are referenced by a user table for a specific extender.

Action: No action is required.

DMB0020I ″<Count>″ files are referenced in tables with table schema ″<name>″ for the ″<extender-name>″ extender.

Cause: The message displays the number of external media files that are referenced by user tables with a specific schema name.

Action: No action is required.

DMB0021I There are ″<count>″ inaccessible files referenced in table ″<table-name>″ for the ″<extender-name>″ extender.

Cause: The message displays the number of external media files that are referenced by a user table for a specific extender, but are inaccessible. The files might have been erased.

Action: No action is required.

DMB0022I There are ″<count>″ inaccessible files referenced by the ″<extender-name>″ extender.

Cause: The message displays the number of external media files that are:

- referenced by user tables in the current database.
- of a specific extender media type (such as video).
- inaccessible; for example, the files might have been erased.

Action: No action is required.

DMB0023I There are ″<count>″ inaccessible files referenced in tables with table schema ″<name>″ for extender ″<extender-name>″.

Cause: The message displays the number of external media files that are referenced by user tables with a specific schema name, but are inaccessible. The files might have been erased. The messages also indicates the number of extenders the tables are enabled for.

Action: No action is required.

DMB0024I The current database is enabled for ″<count>″ extenders.

Cause: The message lists the number of DB2 extenders the current database is enabled for.

Action: No action is required.

DMB0025I Table ″<table-name>″ is enabled for ″<count>″ extenders.

Cause: The message lists the number of DB2 extenders the indicated table is enabled for.

Action: No action is required.

DMB0026I Column ″<column-name>″ in table ″<table-name>″ is enabled for ″<count>″ extenders.

Cause: The message lists the number of DB2 extenders the indicated column is enabled for.

Action: No action is required.

DMB0027I The current database is enabled for extender ″<extender-name>″.

Cause: The message indicates the DB2 extender for which the current database is enabled.

Action: No action is required.

DMB0028I Table ″<table-name>″ is enabled for extender ″<extender-name>″.

Cause: The message indicates the media data type the user table is enabled to hold.

Action: No action is required.

DMB0029I Column ″<column-name>″ in table ″<table-name>″ is enabled for extender ″<extender-name>″.

Cause: The message indicates the media data type the user column is enabled to hold.

Action: No action is required.

DMB0030E The current database cannot be enabled for the ″<extender-name>″ extender. RC = ″<code>″

Cause: Either the database does not exist, or you are not authorized to enable the database.

Action: Make sure the database exists and that you are authorized to enable the database.

DMB0031E The table cannot be enabled for the ″<extender-name>″ extender. RC = ″<code>″

Cause: The database does not exist, or the table is not enabled, or you are not authorized to enable the table.

Action: Make sure the database exists and that both the database and table are enabled for the extender. Make sure you are authorized to enable the table.

DMB0032E The column cannot be enabled for the ″<extender-name>″ extender. RC = ″<code>″

Cause: The column is was not defined using the data type for this extender, or the column does not exist, or the table is not enabled, or you are not authorized to enable the column.

Action: Make sure the column was defined using the right data type. Make sure the table is enabled and you are authorized to enable the column.

DMB0033E You are not authorized to run this command.

Cause: Your user ID does not have the level of authority required to run the command.

Action: Either run the command from another user ID, or get the database administrator to change the level of authority for your current user ID.

DMB0034I The DB2 Extenders Server for database ″<database-name>″ was started successfully.

Cause: The server started successfully for the current database.

Action: No action is required.

DMB0035I The DB2 Extenders Server for database ″<database-name>″ was stopped.

Cause: The server stopped successfully for the current database.

Action: No action is required.

DMB0036E The DB2 Extenders server cannot be started or stopped. The DB2 Extenders server daemon is probably not running. Contact your database administrator.

Cause: An error occurred while starting or stopping the DB2 extenders server. The DB2 extenders server daemon is probably not running.

Action: Please contact your database administrator.

DMB0037E The USE session handle is not valid.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0038E The USE statement handle is not valid.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0039E USE error: ″<error>″

Cause: An internal error has occurred.

Action: Follow the instructions contained in the associated error message and repeat the operation. If the error occurs again, contact IBM service.

DMB0040E SQL error: ″<error>″

Cause: An internal error has occurred.

Action: Follow the instructions contained in the associated error message and repeat the operation. If the error occurs again, contact IBM service.

DMB0041W The current database is re-enabled for the ″<extender-name>″ extender using the newly specified table space.

Cause: When the current database was previously enabled, it used a different table space. The database is now enabled with a new table space for the administrative support tables.

Action: No action is required.

DMB0042E Column ″<column-name>″ in table ″<table-name>″ is not enabled for the ″<extender-name>″ extender.

Cause: The indicated column is not enabled for the extender for which the operation was attempted. For example, you might have tried to disable a column that was not currently enabled for the indicated extender.

Action: Make sure the column is enabled for the extender indicated in the message.

DMB0043I The current database is disabled for the ″<extender-name>″ extender.

Cause: The disable operation was successful.

Action: No action is required.

DMB0044I Table ″<table-name>″ is disabled for the ″<extender-name>″ extender.

Cause: The disable operation was successful.

Action: No action is required.

DMB0045I Column ″<column-name>″ in table ″<table-name>″ is disabled for the ″<extender-name>″ extender.

Cause: The disable operation was successful.

Action: No action is required.

DMB0046E The current database cannot be disabled for the ″<extender-name>″ extender. RC = ″<code>″

Cause: The database does not exist or is not enabled for the extender, or you are not authorized to disable the database.

Action: Make sure the database exists and is enabled for the extender. Make sure you are authorized to disable the database.

DMB0047E The table cannot be disabled for the ″<extender-name>″ extender. RC = ″<code>″

Cause: The table does not exist or is not enabled for the extender, or you are not authorized to disable the table.

Action: Make sure the table exists and is enabled for the extender. Make sure you are authorized to disable the table.

DMB0048E The column cannot be disabled for the ″<extender-name>″ extender. RC = ″<code>″

Cause: The column is not enabled for the extender indicated in the message, so it cannot be disabled for that extender.

Action: Verify the name of the extender and

whether the user column is the one you want to disable.

DMB0049E You are not authorized to run this command.

Cause: Your user ID does not have the level of authority required to run the command.

Action: Either run the application from another user ID, or get the database administrator to change the level of authority for your current user ID.

DMB0050E You do not have "<authority-level>" authority on table "<table-name>".

Cause: The operation requires a level of authority higher than the one of the user ID that made the attempt.

Action: Either perform the operation from the user ID with the right authorization, or get the database administrator to change the level of authority for your current user ID.

DMB0051E Bad media file header.

Cause: The system cannot read or open the header of the media file. Either the file is corrupted, or it is not a media file.

Action: Verify the file is a media file and is not corrupted.

DMB0052I The DB2 Extenders server for database "<database-name>" was started successfully.

Cause: The server started successfully.

Action: No action is required.

DMB0053I The DB2 Extenders server for database "<database-name>" was stopped successfully.

Cause: The server stopped successfully.

Action: No action is required.

DMB0054E The DB2 Extender server cannot connect to the database or allocate a DB2 statement handle. The DB2 Extender server for database "<database-name>" is probably not running.

Cause: The DB2 extenders server cannot connect to the database or allocate a DB2 statement handle. The DB2 extenders server for the database is probably not running.

Action: Make sure the DB2 extender server for the database is running. If it is not, either start the specific extender server for the database, or ask your system administrator to restart the extender services.

DMB0055I The "command-name" command completed successfully.

Cause: The command completed successfully.

Action: No action is required.

DMB0056E An unexpected token "<token>" was found following "<keyword>". Expected tokens can include: <extendername>.

Cause: The command expected the name of a DB2 extender instead of the token indicated in the message.

Action: Follow the syntax of the command and try again.

DMB0057E The table space "<table-space-name>" is not valid.

Cause: The table space in the message does not exist.

Action: Verify the name of the table space and whether it exists.

DMB0058I "<Count>" files are referenced by the "<extender-name>" extender.

Cause: The message displays the number of external media files that are referenced by a specific extender.

Action: No action is required.

DMB0059E ″<Name>″ is not a valid name for an DB2 extender. Valid extender names include ″<extender-name,>″ DB2VIDEO, DB2AUDIO, and DB2IMAGE.

Cause: The extender name is misspelled.

Action: Verify the extender name.

DMB0060E The correct syntax for ″<function>″ is: ″<syntax>″

Cause: The syntax of the command you entered is wrong.

Action: Follow the syntax as described in the message.

DMB0061E The table name ″<name>″ that follows ″<keyword>″ is not valid.

Cause: The command expected the name of a table.

Action: Follow the syntax of the command and try again.

DMB0062E The column name ″<name>″ that follows ″<keyword>″ is not valid.

Cause: The command expected the name of a column.

Action: Follow the syntax of the command and try again.

DMB0064E The system does not recognize the token ″<token>″ that follows ″<keyword>″.

Cause: The command expected something other than the token indicated in the message.

Action: Follow the syntax of the command and try again.

DMB0065E The user ID ″<identifier>″ that follows ″<keyword>″ is not valid.

Cause: The command expected a valid user ID.

Action: Verify the user ID you want and try again.

DMB0066E The password ″<password>″ that follows ″<keyword>″ is not valid.

Cause: The command expected a valid user ID instead of the token indicated in the message.

Action: Verify the password and try again.

DMB0067E The command you entered is not valid.

Cause: The name of the command was misspelled or the syntax was wrong.

Action: Follow the syntax of the command and try again.

DMB0068E Metadata table does not exist.

Cause: The function tried to use an administrative support (metadata) table that should exist for the data object. The metadata table might have been corrupted or erased.

Action: Check the name and verify the existence of the metadata table. If the metadata tables were accidentally erased or corrupted, disable and then re-enable the data object.

DMB0069E DBname ″<name>″ invalid.

Cause: A database with that name does not exist.

Action: Check the name and verify the existence of the database.

DMB0070E Handle not valid.

Cause: The handle value you passed in your application might be corrupted.

Action: Verify your application to make sure the extender handle values are not modified.

DMB0071E Can't connect to "<database-name>".

Cause: The DB2 extender server for the database might not be started.

Action: Check the status of the server. If the server is not running, restart it using the START SERVER command on the DMB command line.

DMB0072E UDF SQL server can't disconnect from DB.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0073E USE session handle not valid.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0074E USE statement handle not valid.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0075E Specify a file name.

Cause: The operation requires a media file name.

Action: Enter the name of a media file.

DMB0076E Can't open import file.

Cause: The import file is either missing or corrupted.

Action: Verify the name and existence of the import file.

DMB0077E Can't open/read content file.

Cause: The extender handle points to a file that does not exist or is corrupted. The file has become inaccessible to the extender.

Action: Use the FILENAME UDF to find the name of the file, or verify the existence of the content file.

DMB0078E Can't create export file.

Cause: The export file is either missing or corrupted.

Action: Verify the name and existence of the export file.

DMB0079E Can't copy BLOB to file.

Cause: The file cannot accept the BLOB. There might not be enough storage space to accommodate the BLOB.

Action: Compare the size of the BLOB to the available storage, and increase storage if necessary.

DMB0080E Can't write to file.

Cause: The file is corrupted or does not exist, or the name is misspelled.

Action: Verify the name and existence of the file.

DMB0081E Offset or size invalid.

Cause: The operation did not find the expected data in the data structure. Either the size of the field or the offset is incorrect.

Action: Verify the offset and the size of the field.

DMB0082E Can't build handle.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0083E "<extender-name>" and "<extender-name>" incompatible.

Cause: The two extenders specified in the message are not compatible in this usage. The

insert operation, by either full or subselect, is not valid.

Action: Make sure your target object is enabled for the same extender for which the source object is enabled.

DMB0084E Import request invalid: filename, content, storage type.

Cause: The import operation failed. Either the file name, the content, or the storage type was not valid.

Action: Check the data and try again.

DMB0085E The update request is not valid: filename, content, storage type.

Cause: The update operation failed. Either the file name, the content, or the storage type was not valid.

Action: Check the data and try again.

DMB0086E Requested size too large.

Cause: The size you requested is larger than the maximum blob size for the UDF.

Action: Request a smaller size.

DMB0087E File name invalid.

Cause: There is no file with that name.

Action: Verify the name and existence of the file.

DMB0088E Handle value is NULL.

Cause: The UDF was expecting a non-null handle.

Action: Make sure the application gets a valid handle and passes it to the UDF.

DMB0089E Handle value doesn't exist.

Cause: The handle passed to the UDF is not valid.

Action: Make sure the application passes a valid handle.

DMB0090E Data truncated.

Cause: The file or buffer was too small to accept the data.

Action: Increase the size of the file or buffer.

DMB0091W Content already in file.

Cause: The file already has content. The content will be overwritten.

Action: No action is required.

DMB0092E The insert operation attempted for column ″<column-name>″ is not valid. The column is enabled for the ″<extender-name>″ extender.

Cause: The data type of the data being inserted is different from the extender for which the column is enabled.

Action: Make sure your target object is enabled for the same extender for which the source object is enabled.

DMB0093E The update operation attempted for column ″<column-name>″ is not valid. The column is enabled for the ″<extender-name>″ extender.

Cause: The data type of the data being updated is different from the extender for which the column is enabled.

Action: Make sure your target object is enabled for the same extender for which the source object is enabled.

DMB0094I Table ″<table-name>″ does not exist.

Cause: The system cannot find a table with that name. It might exist in another database.

Action: No action is required.

DMB0095W The table ″<table-name>″ is not enabled for the ″<extender-name>″ extender.

Cause: The table is not enabled for the extender.

Action: No action is required.

DMB0096W Column ″<column-name>″ in table ″<table-name>″ was not enabled for the ″<extender-name>″ extender.

Cause: The system expected the column to be enabled.

Action: No action is required

DMB0097W The current database is not enabled for the ″<extender-name>″ extender.

Cause: The system expected the database to be enabled.

Action: Enable the database for the extender indicated in the message.

DMB0098E The user does not have ″<authority-level>″ authority on table ″<table-name>″.

Cause: The operation requires a level of authority higher than the one of the user ID that made the attempt.

Action: Either perform the operation from the user ID that owns the table, or ask the database administrator to change the level of authority for your current user ID.

DMB0099E Can't commit transaction.

Cause: The extender server for the current database might be stopped.

Action: Check the status of the server. If the server is not running, restart it using the START SERVER command on the db2ext command line.

DMB0100E ″<name>″ is not a valid table name.

Cause: No table with that name exists.

Action: Verify the name and existence of the table and try again.

DMB0101E Invalid NULL parameter.

Cause: The command was expecting a non-null parameter.

Action: Check the syntax and try again.

DMB0102E Invalid storage type.

Cause: For the DB2 extenders, the storage type indicates where the media data is stored.

Action: Specify 0 to indicate external (in a file) and 1 to indicate external (in the database).

DMB0103E Unsupported format.

Cause: DB2 extenders do not support the format of this object.

Action: Convert the object to a supported format.

DMB0104E Video content buffer too small.

Cause: The video clip is too big for the buffer allocated for it.

Action: Allocate a bigger buffer.

DMB0105E MPEG1 header invalid.

Cause: The header of the MPEG1 file is missing or corrupt.

Action: Verify the file is a MPEG1 file.

DMB0106E AVI header invalid.

Cause: The header of the AVI file is missing or corrupt.

Action: Verify the file is an AVI file.

DMB0107E Export environment not set up.

Cause: In the DB2 extenders, the environment variables for the export environment are not set properly.

Action: Make sure the environment variables are set properly, as described in "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

DMB0108E Import environment not set up.

Cause: In the DB2 extenders, the environment variables for the import environment are not set properly.

Action: Make sure the environment variables are set properly, as described in "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

DMB0109E Can't resolve import file.

Cause: There is no import file with that name.

Action: Verify the name and existence of the file and make sure the environment variables are set properly, as described in "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

DMB0110E Can't resolve export file.

Cause: There is no export file with that name.

Action: Verify the name and existence of the file and make sure the environment variables are set properly, as described in "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

DMB0111E Store environment not set up.

Cause: The environment variables for the store environment are not set properly,

Action: Make sure the environment variables are set properly, as described in "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

DMB0112E Can't resolve store file.

Cause: There is no store file with that name.

Action: Verify the name and existence of the file and make sure the environment variables are set properly, as described in "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

DMB0113E Can't open import file.

Cause: The file might be locked by someone else, or the file is missing or corrupt.

Action: Verify the name, existence, and status of the file, and your authorization level.

DMB0114E Can't open export file.

Cause: The file might be locked by someone else, or the file is missing or corrupt.

Action: Verify the name, existence, and status of the file, and your authorization level.

DMB0115E Can't open store file.

Cause: The system is trying to write a file but the file already exists. The server does not have the authority to overwrite the file.

Action: Verify the name, existence, and status of the file, and your authorization level.

DMB0116E Can't create temporary file.

Cause: There might not be enough storage space to create the temporary file.

Action: Make sure the temporary environment variables for the extender are set properly. Increase the storage if necessary.

DMB0117E Temporary environment not set up.

Cause: The environment variables for the temporary environment are not set properly,

Action: Make sure the environment variables are set properly, as described in "Appendix A.

Setting Environment Variables for DB2 Extenders" on page 551.

DMB0118E Can't open temporary file.

Cause: The temporary file might have been overwritten or corrupted.

Action: Make sure the environment variables are set properly, as described in "Appendix A. Setting Environment Variables for DB2 Extenders" on page 551.

DMB0119I The dmbsrv server is starting for ″<name>″ with ″<count>″ connections.

Cause: The message indicates how many connections the server starts with.

Action: No action is required.

DMB0120E The dmbsrv server failed to start for ″<name>″ with ″<count>″ connections.

Cause: DB2 might not be started yet, or the database might not exist, or your system might have run out of licensed connections.

Action: Make sure DB2 is started and the database exists. If the problem persists, contact IBM to get more licences.

DMB0121I The dmbsrv server is started for ″<name>″ with ″<count>″ connections.

Cause: The message indicates how many connections the server starts with.

Action: No action is required.

DMB0122I The dmbssd server is ready.

Cause: The server is ready to run your application.

Action: No action is required.

DMB0129E Invalid operation: ″<operation-name>″.

Cause: No command or API exists with that name.

Action: Verify the command or API and try again.

DMB0130E Column ″<column-name>″ failed to be bound to the SQL statement.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0131E SQL prepare statement failed.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0132E SQL set parameter failed.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0133E SQL execute statement failed.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0134E File format conversion failed.

Cause: The format of the stored multimedia data is not support for format conversion.

Action: You cannot convert the format of this file.

DMB0135E Can't open/read thumbnail.

Cause: The thumbnail file might be missing or corrupted.

Action: Verify the name, existence, and integrity of the thumbnail file.

DMB0136E Can't find bind file.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0137E Can't connect to DB "<database-name>"

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0138E Can't free an SQL statement.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0139E The feature name "<name>" that follows "<keyword>" is not valid.

Cause: The Image Extender expected a valid feature name in this command.

Action: Try the command again with a valid feature name. Valid feature names include:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

DMB0141E The qualifier "<identifier>" that follows "<keyword>" is not valid.

Cause: The system cannot identify the qualifier in the command.

Action: Check the qualifier and try again.

DMB0142E No catalog was opened.

Cause: In the DB2 extenders, the current command needs a QBIC catalog to be opened.

Action: Open the QBIC catalog with the OPEN QBIC CATALOG command.

DMB0143I In the QBIC catalog for column "<column-name>" in table "<table-name>", auto-cataloging has been set to "<status>". There are "<count>" features:

Cause: The message indicates the number of features defined in the QBIC catalog for a specific image column, and whether auto-cataloging is turned on.

Action: No action is required.

DMB0145E The query handle is not valid.

Cause: The query handle you used in the API call is not valid.

Action: Check your application to see if you are obtaining the correct query handle.

DMB0146E The feature class name "<feature-class>" is not valid.

Cause: There is no feature class with that name. Valid feature names include:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Action: Correct the name of the feature and try again.

DMB0147E The feature class name "<feature-class>" is either missing or not valid.

Cause: Valid feature names include:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Action: Correct the name of the feature and try again.

DMB0148E Feature ″<feature-name>″ is already a member of the query.

Cause: The query already supports the feature indicated in the message.

Action: No action is required.

DMB0149E Feature ″<feature-name>″ is not a member of the query.

Cause: The query does not include the specified feature name.

Action: Use the QbQueryAddFeature API to add the feature to the query before calling other APIs that access the feature.

DMB0150E The system cannot allocate memory.

Cause: The system was unable to allocate memory required to support the attempted operation.

Action: Verify that your system has enough memory to complete the operation.

DMB0151E The pointer to the return value is NULL.

Cause: The API call did not complete successfully because the pointer to a return value must not be NULL.

Action: Make sure that valid parameters are supplied to the API call and the syntax is followed correctly.

DMB0152E The pointer to the list return value is NULL.

Cause: The API call did not complete successfully because the pointer to a return value must not be NULL.

Action: Make sure that valid parameters are supplied to the API call and the syntax is followed correctly.

DMB0153E The scope parameter is reserved and must be 0.

Cause: The parameter is reserved for future use.

Action: Set the scope to 0.

DMB0154E The pointer to the feature class name is not valid.

Cause: The API call expected a valid pointer to the input feature class name.

Action: Make sure that valid parameters are supplied to the API call and the syntax is followed correctly.

DMB0155I A buffer size of zero was passed to the ″<function-name>″ function.

Cause: The API call needs the buffer to return information.

Action: No action is required.

DMB0156E The QbImageSource pointer is NULL.

Cause: A NULL value indicates that the structure should not be changed.

Action: No action is required.

DMB0157E The QbImageSource type ″<type>″ is not valid.

Cause: The data structure referenced by this DB2 extender API is of the wrong data type.

Action: The data type of the structure should be QbImageSource.

DMB0159E The pointer to the QbImageSource image buffer is NULL.

Cause: The API call expected a pointer to be returned.

Action: Check your application to see if the API call and the buffer is specified correctly.

DMB0160I The length of the image buffer or file is zero.

Cause: The length is zero.

Action: No action is required.

DMB0161E The pointer to the table and/or column name is NULL.

Cause: The API call expected a pointer to be supplied.

Action: Check your application to see if the input to the API call is specified correctly.

DMB0162I You set requestedHits to zero.

Cause: With requestedHits set to zero, you get nothing back.

Action: No action is required.

DMB0163I That function is not yet supported.

Cause: That function is not yet supported.

Action: No action is required.

DMB0164E The system cannot process the query (<query-name>).

Cause: An error occurred when the query was created.

Action: Check the input to the command or API and try again.

DMB0165E The system cannot execute the query (<query-name>).

Cause: An error occurred when the query was created.

Action: Check the input to the command or API and try again.

DMB0166E A statement error was found in ″<name>″ while executing ″<name>″: ″<error>″

Cause: An internal IBM error occurred.

Action: Please contact your database administrator.

DMB0167E An error occurred while reading QbGenericImageDataClass (<error>).

Cause: An internal IBM error occurred.

Action: Please contact your database administrator.

DMB0168E A query's feature ″<feature-name>″ was not set prior to search.

Cause: The query did not run because it had no feature assigned to it.

Action: Add a feature to the query using either the QbAddFeature API or the ADD QBIC FEATURE command.

DMB0169E The following error occurred in the Call-Level Interface: ″<error>″.

Cause: CLI error.

Action: Follow the directions in the message text.

DMB0170E Query name ″<query-name>″ is already in use.

Cause: Another query exists with that name.

Action: Select another name.

DMB0171E Query name ″<query-name>″ has not been stored.

Cause: After creating the query, the system could not store it.

Action: Make sure you have write authority and enough storage to store the query.

DMB0172E SQL Error: ″<error>″.

Cause: An internal error has occurred.

Action: Follow the instructions contained in the associated error message and repeat the operation. If the error occurs again, contact IBM service.

DMB0173E The catalog is open, but for read-only: ″<catalog-name>″.

Cause: You cannot update the catalog because someone else already opened the catalog in write mode, or you do not have write authority for it.

Action: Wait until the other user is finished, run the application from another user ID, or get the database administrator to change the level of authority for your current user ID.

DMB0174E A system error occurred: ″<error>″.

Cause: An internal IBM error occurred.

Action: Follow the instructions contained in the associated error message and repeat the operation. If the error occurs again, contact IBM service.

DMB0175I Images were not found: ″<information>″.

Cause: No images were found that matched the query. The database might be empty.

Action: No action is required.

DMB0176I The column already has a QBIC catalog: ″<table-name column-name>″.

Cause: Another catalog exists with that name.

Action: No action is required.

DMB0177E The system cannot open the catalog; the error message is: ″<error>″.

Cause: The catalog was corrupted.

Action: Follow the instructions in the message text.

DMB0178E The system cannot delete the catalog; the error message is: ″<error>″.

Cause: Either the catalog does not exist, or it was corrupted.

Action: Verify the name and existence of the catalog and try again.

DMB0179E The catalog handle is not valid: ″<error>″.

Cause: The catalog handle you used in the API call is not valid.

Action: Check your application to see if you are obtaining the correct catalog handle.

DMB0180I Access to catalog is denied: ″<error>″.

Cause: Access is denied.

Action: No action is required.

DMB0181I Catalog is in use ″<error>″.

Cause: Another operation is using this catalog.

Action: No action is required.

DMB0184I Tracing has already been started:

Cause: Tracing has already been started.

Action: No action is required.

DMB0185I Tracing has not been started yet.

Cause: Tracing has not been started yet.

Action: No action is required.

DMB0186I Tracing was turned on at ″<time>″ from directory ″<directory-name>″. The trace file is ″<file-name>″. ″<Count> bytes of trace data have been written.

Cause: Tracing is on.

Action: No action is required.

DMB0187E Communication cannot be established because the system cannot open file ″<file-name>″ for writing.

Cause: Either you are not the owner of the current instance described by environment variable DB2INSTANCE, or the environment variables such as DB2MMTOP are not set correctly.

Action: Log with the user ID that owns the instance. Verify that the environment variables are set correctly.

DMB0188I An error occurred when creating the trace daemon: ″<error>″

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0189I Tracing has already been successfully started:

Cause: Tracing has already been started.

Action: No action is required.

DMB0190E Tracing cannot be started.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0191E Environment variable ″<name>″ needs to be set.

Cause: The system configuration is not correct.

Action: Set the variable and try again.

DMB0192I Tracing has been successfully turned off.

Cause: Tracing is off.

Action: No action is required.

DMB0193E The system cannot write to file ″<file-name>″.

Cause: You do you have write authority for the directory of the specified file.

Action: Please contact your database administrator to get authority.

DMB0194E The system cannot read from file ″<file-name>″.

Cause: Either the file does not exist or you do not have read authority for the file.

Action: Make sure the file exists and that you have read authority for the file.

DMB0198E The trace code ″<code>″ in the input file is unknown. The input file might be corrupted.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0199E You do not have ″<authority-level>″ authority for any of the referenced tables.

Cause: Your user ID does not have the level of authority required for the operation.

Action: Either perform the operation from another user ID, or get the database administrator to change the level of authority for your current user ID.

DMB0200W You do not have "<authority-level>" authority for at least one of the referenced tables.

Cause: Your user ID does not have the level of authority required for some tables.

If you are listing referenced files, the files that are listed are referred to by tables for which you have Select authority. If there are tables on your system for which you do not have Select authority, the files referenced by them are not listed.

If you are reorganizing metadata, the system only reorganized metadata for tables for which you have Control authority.

Action: To get all the files, either perform the operation from another user ID, or get the database administrator to change the level of authority for your current user ID.

DMB0201I A feature with that name already exists: "<feature-name>".

Cause: A feature with that name is already included in the QBIC catalog.

Action: No action is required.

DMB0202E The feature name is not valid: "<feature-name>".

Cause: There is no feature class with that name. Valid feature names include:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Action: Correct the name of the feature and try again.

DMB0203E The feature was not found: "<feature-name>".

Cause: There is no feature class with that name, or the feature class is not included in the QBIC catalog. Valid feature names include:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Action: Correct the name of the feature and try again.

DMB0204E The column is not enabled for DB2IMAGE: "<column-name>".

Cause: The column is not enabled for the Image Extender.

Action: Make sure the column is enabled for the DB2 Image Extender.

DMB0205E No catalog found for "<table-name column-name>".

Cause: There is no QBIC catalog associated with the specified column.

Action: Create a QBIC catalog for the column before performing any other QBIC operations.

DMB0206W The specified column is not enabled for any extender.

Cause: The column might not exist or its data type might not be compatible with the extenders.

Action: Verify the column has been defined using the correct data type.

DMB0207E Can not overwrite the file.

Cause: The file already exists but the EXPORT UDF cannot overwrite it.

Action: Export the file to a different file name or allow the EXPORT UDF to overwrite the file.

DMB0208E sqlcode=<code> clistate=<code>.

Cause: An internal error has occurred.

Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0209E Invalid audio header.

Cause: The header of the audio file is missing or corrupt.

Action: Verify that the format of the audio file is supported by DB2 extenders.

DMB0211W File exists w/o overwrite.

Cause: The specified target file already exists and is not overwritten.

Action: No action is required.

DMB0212E The resultType parameter is reserved and must be 0.

Cause: The parameter is reserved for future use.

Action: Set the resultType to 0.

DMB0214E The pointer to the query name is not valid.

Cause: The API call expected a valid pointer to the input query name.

Action: Make sure that valid parameters are supplied to the API call and the syntax is followed correctly.

DMB0352E Command line environment not initialized.

Cause: The command line environment is not initialized to run the db2ext command-line processor. (This message applies only to Windows NT and Windows 95 environments.)

Action: Issue the db2cmd command to open a DB2CLP window, then issue the db2ext command to run the db2 command-line processor in that window.

DMB0353E Cannot communicate with db2ext command-line processor's background process.

Cause: The background process for the db2ext command-line processor is running, but the db2ext command-line processor cannot communicate with it.

Action: Try issuing the db2ext command in a different window.

DMB0354E Cannot start db2ext command-line processor's background process.

Cause: The background process for the db2ext command-line processor is running, but the db2ext command-line processor cannot communicate with it.

Action: Check that the executable module for the background process (db2extb or db2extb.exe) exists, and its directory is in the PATH environment variable.

DMB0355E db2ext command-line processor's background process timed out.

Cause: The background process for the db2ext command-line processor started successfully, but the db2ext command-line processor cannot communicate with it within the allowed time limit.

Action: Try issuing the db2ext command in a different window.

DMB0356E Cannot communicate with the db2ext command-line processor's background process.

Cause: The db2ext command-line processor sent a request to its background process, but the request was not received.

Action: Make sure the background process for the db2ext command-line processor is still running.

DMB0357E db2ext command-line processor's background process is not responding.

Cause: The db2ext command-line processor sent a request to its background process, but the background process did not respond within the allowed time limit.

Action: Make sure that the background process for the db2ext command-line processor is still running.

DMB0359E The db2ext command-line-processor background process request queue or input queue was not created within the timeout period.

Cause: The background process for the db2ext command-line processor did not create message queues within the allowed time limit. (This message applies only to UNIX environments.)

Action: Make sure that the disk on which the DB2 instance home directory resides is not full (the background process needs this home directory to create a file for message queues). If the disk is not full, check whether you have started too many db2extb processes. This is possible if you are running the db2ext command-line processor in many windows. A background process is started in a window the first time you issue a db2ext command-line processor request in command mode. Make sure to issue the command `db2ext terminate` to terminate the db2ext command-line processor when you no longer need it. Message queues for the backend process are deleted only if you issue the `terminate` command.

DMB0361E Column or table not enabled.

Cause: An import UDF was specified, but the specified table column is not enabled for the extender.

Action: Enable the table column and retry.

DMB0363E Missing table and column name.

Cause: An update UDF was invoked, but a table was not specified.

Action: Specify a table and retry.

DMB0364E Extender "<extender-name>" has been previously defined for the table space "<tablespace-name>".

Cause: The specified database, table, or column has already been enabled for the extender using a different tablespace than the one specified.

Action: Check that the table space specification is correct.

DMB0365E You don't have CONTROL privilege on "<metadata-table-name>" and "<metadata-table-name >" which are the metadata tables for "<schema-name>"."<table-name>".

Cause: Your request was denied because you do not have the required CONTROL privilege on the metadata tables for the specified user table.

Action: Have your database administrator grant you CONTROL privilege on the metadata tables.

DMB0366E Expected feature name.

Cause: A feature name is expected in the query string.

Action: Correct the query string and try again.

DMB0367E Expected color|color histogram|file.

Cause: Either "color", "histogram", or "file" is expected in the query string.

Action: Correct the query string and try again.

DMB0368E Expected ','.

Cause: A ',' is expected in the query string.

Action: Correct the query string and try again.

DMB0369E File is not valid.

Cause: The file specified in the query string is invalid.

Action: Correct the query string and try again.

DMB0370E Expected filename.

Cause: A filename is expected in the query string.

Action: Correct the query string and try again.

DMB0371E Expected server|client.

Cause: Either “server” or “client” is expected in the query string.

Action: Correct the query string and try again.

DMB0372E Expected '('.

Cause: A '(' is expected in the query string.

Action: Correct the query string and try again.

DMB0373E Expected ')'.

Cause: A ')' is expected in the query string.

Action: Correct the query string and try again.

DMB0374E Expected percentage.

Cause: The percent value is expected in the query string.

Action: Correct the query string and try again.

DMB0375E Expected color.

Cause: The red, green, and blue values are expected in the query string.

Action: Correct the query string and try again.

DMB0376E Expected '='.

Cause: An '=' is expected in the query string.

Action: Correct the query string and try again.

DMB0377E Expected '<'.

Cause: An '<' is expected in the query string.

Action: Correct the query string and try again.

DMB0378E Expected '>'.

Cause: An '>' is expected in the query string.

Action: Correct the query string and try again.

DMB0379E Expected 'and'.

Cause: An 'and' is expected in the query string.

Action: Correct the query string and try again.

DMB0380E Expected weight.

Cause: A weight is expected in the query string.

Action: Correct the query string and try again.

DMB0381E Feature not set.

Cause: The feature has not been added to the QBIC catalog.

Action: Add the feature to the QBIC catalog, and recatalog the images.

DMB0382E Could not build query.

Cause: The extender server for the current database might be stopped.

Action: Check the status of the server. If the server is not running, restart it using the START SERVER command on the db2ext command line.

DMB0383E Could not execute query.

Cause: The extender server for the current database might be stopped.

Action: Check the status of the server. If the server is not running, restart it using the START SERVER command on the db2ext command line.

DMB0384E Could not get next item.

Cause: End of the list has been reached.

Action: Make sure that your application is not attempting to retrieve an item beyond the end of the list.

DMB0386E Can't collocate with user data

Cause: The SQL API sqluihsh() has returned a non-zero return code.

Action: Retry. If the problem persists, call IBM support.

DMB0387E The nodegroup for the specified tablespaces is different from the nodegroup of the user table.

Cause: One or more of the tablespaces (that is, for the metadata table, index, or BLOB) passed as input for enabling a table is defined on a nodegroup that is different from the one the user table is defined on.

Action: Use tablespaces that are defined on the same nodegroup as the user table to be enabled.

DMB0388E The regular, long or index tablespaces are not defined on the same nodegroup.

Cause: One or more of the tablespaces (that is, for the metadata table, index, or BLOB) passed as input for enabling a database is not defined on the same nodegroup as the other tablespaces.

Action: Use tablespaces that are defined on the same nodegroup.

DMB0389W The nodegroup for the specified tablespaces does not include all of the partition servers.

Cause: The tablespaces passed as input are defined on a nodegroup that does not include all partition servers.

Action: No action is required. However, the import and update UDFs will perform more efficiently if the tablespaces are defined on a nodegroup that covers all the partition servers. This is especially true if the extender applications store media content in BLOB format.

DMB0391I This command can be run only when a DB2 UDB client is accessing a DB2 UDB server.

Cause: Either the db2ext command-line processor is not connected to a DB2 UDB server, or the db2ext command-line processor has not been invoked by a DB2 UDB client. For example, the command START SERVER is valid only if the db2ext command-line processor is connected to a DB2 non-Extended Enterprise Edition server.

Action: Do not issue this command in the current client/server configuration.

DMB0392I The command can be run only when a DB2 UDB client is accessing a DB2 UDB Extended Enterprise Edition server. For example, the command DISCONNECT SERVER is valid only if the db2ext command-line processor is connected to a DB2 Extended Enterprise Edition server.

Cause: Either the db2ext command-line processor is not connected to a DB2 UDB Extended Enterprise Edition server, or the db2ext command-line processor has not been invoked from a DB2 UDB client.

Action: Do not issue this command in the current client/server configuration.

DMB0402E Option "<option-name>" for command "<command-name>" is valid only if the application is connected to a DB2 "<server-type>" server.

Cause: The specified parameter is not valid because the db2ext command-line processor is not connected to the type of server that supports that option. For example, the command GET SERVER STATUS can be specified with the parameter NODENUM <nodenum> only if the db2ext command-line processor is connected to a DB2 Extended Enterprise Edition server.

Action: Do not issue this command-parameter

combination in the current client/server configuration.

DMB0411E Invalid base port

Cause: An invalid TCP/IP port number was entered as the base port during instance creation.

Action: The correct syntax is dmbicrt -r:base_port,end_port -t:base_port,end_port. Correct the parameter and retry the command.

DMB0412E Invalid end port

Cause: An invalid TCP/IP port number was entered as the end port during instance creation.

Action: The correct syntax is dmbicrt -r:base_port,end_port -t:base_port,end_port. Correct the parameter and retry the command.

DMB0413E Unable to resolve the DB2 Extenders installation path.

Cause: The instance creation program could not find a value for the environment variable "DMBPATH."

Action: Set the variable "DMBPATH" and retry the application.

DMB0414E Unable to resolve the computer hostname.

Cause: An internal error was encountered while trying to resolve the computer's name.

Action: Contact IBM support.

DMB0415E Unable to resolve the node number for this machine.

Cause: The machine on which the instance creation is being run is not listed in the file "db2nodes.cfg."

Action: Add the machine to the "db2nodes.cfg" and retry the application.

DMB0416E This program must be executed by the root. Unable to continue.

Cause: The user ID under which the program is being run does not have root authority.

Action: Logon as the root and retry the application.

DMB0417E This program must be executed by a user with administrator authority. Unable to continue.

Cause: The user ID under which the program is being run does not have administrator authority.

Action: Logon with a user ID that has administration authority, and retry the application.

DMB0418E Unable to get information about user: "<userid>".

Cause: An internal error occured when trying to get user information about the user ID associated with the instance being created.

Action: Ensure that there is a valid user ID with the same name as the instance being created, and retry the application.

DMB0419E Unable to create AIV Extenders directory "<directory_name>. Return Code = <code>

Cause: An error occured while trying to create the specified directory. The return code represents the error returned from the operating system.

Action: Ensure that the file system/drive specified in the directory name exists and that permissions allow a directory to be created.

DMB0420E Unable to create link for AIV Extenders directory "<directory_name>". Return Code = <code>

Cause: An error occured while trying to create the specified symbolic link. The return code

represents the error returned from the operating system.

Action: Ensure that the file system/drive specified in the directory name exists and that permissions allow a link to be created.

DMB0421E Unable to open file: ″<file_name>″. Return Code = <code>

Cause: An error occured while trying to open the specified file. The return code represents the error returned from the operating system.

Action: Ensure that the file exists and that permissions allow the file to be opened.

DMB0422E Unable to write to file: ″<file_name>″. Return Code = <code>

Cause: An error occurred while trying to write to the specified file. The return code represents the error returned from the operating system.

Action: Ensure that the file exists and that permissions allow the file to be written to.

DMB0424E Unable to find ″db2nodes.cfg″ file.

Cause: The DB2 file ″db2nodes.cfg″ could not be located.

Action: Ensure that the correct version of DB2 UDB Extended Enterprise Edition has been installed and retry the application.

DMB0426E Error: ″<error_code>″ opening key ″<registry_key>″.

Cause: An error occurred while trying to open the specified registry key.

Action: Record the return code and contact IBM support.

DMB0427E The variable ″<variable>″ was not set in the profile registry.

Cause: The specified value was not found in the Windows NT registry.

Action: Ensure that the name of a valid DB2 Extender variable was specified.

DMB0430E Unable to find DB2 registry values

Cause: The registry values used by DB2 could not be found.

Action: Ensure that the correct version of DB2 UDB Extended Enterprise Edition is installed and retry the application.

DMB0431E Unable to create Extender Registry Key: ″<registry_key>″.

Cause: An internal error occured while trying to create an extender registry key.

Action: Contact IBM support.

DMB0432E Unable to set value for extender registry key: ″<registry_key>″.

Cause: An internal error occurred while trying to set an extender registry key value.

Action: Contact IBM support.

DMB0435E Unable to access control file ″<control_file>″.

Cause: The specified control file could not be found.

Action: Contact IBM support.

DMB0443E Unable to open directory ″<directory_name>″. Return = <code>

Cause: An error occured while trying to open the specified directory. The return code represents the error returned from the operating system.

Action: Ensure that the file system/drive specified in the directory name exists and that permissions allow a directory to be opened.

DMB0449W -q:datapath is required for AIV extender instance creation.

Cause: The '-q' parameter was not specified when trying to create an AIV Extender instance.

Action: Specify the parameter and retry the application.

DMB0450W One or more of the specified "<port>" ports are already in use.

Cause: A port was specified for use by DB2 Extenders that is already listed in the services file as being in use.

Action: Specify a port or ports that are not in use and retry the application.

DMB0452E Node number "<node_num>" was not found in the "db2nodes.cfg" file.

Cause: The node number of this machine was not found in the "db2nodes.cfg" file.

Action: Add the node number to the "db2nodes.cfg" file and retry the application.

DMB0460W Unable to determine if TCP/IP ports are available.

Cause: An error occured while trying to verify if the specified TCP/IP ports are already in use.

Action: Ensure that the ports specified are not listed in the services file as being in use by another application.

DMB0462E Unable to initialize this node. Return code = <code>.

Cause: Extender startup encountered an error while trying to initialize the current node.

Action: Contact IBM support.

Diagnostic tracing

The DB2 extenders include a trace facility that records extender server activity. You should use the trace facility only under instruction of IBM service personnel.

The trace facility records information in a server file about a variety of events, such as entry to or exit from a DB2 extender component or the return of an error code by a DB2 extender component. Because it records information for many events, the trace facility should be used only when necessary, for example, when you are investigating error conditions. In addition, you should limit the number of active applications when using the trace facility. Limiting the number of active applications can make it easier to isolate the cause of a problem.

Use the DMBTRC command to control tracing. You can issue the command from a command line on an OS/2 server, AIX server, or Windows NT server. You must have SYSADM, SYSCTRL, or SYSMINT authority to issue the command.

Use the DMBTRC command to:

- Start tracing

- Stop tracing
- Reformat trace information to make it more readable
- Show trace status

Start tracing

You can start tracing by entering the command:

dmbtrc on *path*

where *path* is the path of a server file that will contain the trace information.

For example, the following command starts tracing:

```
dmbtrc on /tmp/trace.txt
```

Stop tracing

You can stop tracing by entering the command:

dmbtrc off

Reformat trace information

Trace information is recorded in binary format. You can reformat the information and make it more readable by entering the following command:

`dmbtrc format` *input_file output_file*

where *input_file* is the file that contains the trace information in binary format, and *output_file* is the file that will contain the reformatted information. The *output_file* parameter is optional; if you do not specify it,the reformatted information is displayed on the screen.

For example, the following command copies trace information from one file to another and then reformats it:

```
dmbtrc format /tmp/trace.txt /tmp/fmttrace.txt
```

Show trace status

Use the command:

```
dmbtrc info
```

to show the following trace status information:

- Trace facility on or off
- Path of the file that contains the trace information

Part 5. Appendixes

Appendix A. Setting Environment Variables for DB2 Extenders

The DB2 extenders give you flexibility in how you specify file names when you store, retrieve, or update image, audio, or video objects. You also have flexibility in how you specify programs to display or play image, audio, and video objects retrieved from a database table.

How environment variables are used to resolve file names

Although you can specify a fully qualified file name, (that is, a complete path followed by the file name) for store, retrieve, and update operations, it's preferable to specify a relative file name. In AIX, HP-UX, or Solaris, a relative file name is any file name that does not begin with a slash; in OS/2 and Windows, a relative file name is any file name that does not begin with a drive letter followed by a colon and backslash.

If you specify a relative file name, the extenders will use the directory specifications in various client and server environment variables to resolve the file name. This allows files to be moved in a client/server environment without changing the file name. A fully qualified file name would have to be changed every time a file is moved.

Table 17 lists and describes environment variables that you can set for use by the Image, Audio, and Video Extenders in resolving file names.

Table 17. Environment variables for DB2 extenders

Image Extender	Audio Extender	Video Extender	Description
Server environment variables			
DB2IMAGEPATH	DB2AUDIOPATH	DB2VIDEOPATH	Used to resolve source file name for store, retrieve, and update operations from a server file.
DB2IMAGESTORE	DB2AUDIOSTORE	DB2VIDEOSTORE	Used to resolve target file name for store and update operations to a server file.
DB2IMAGEEXPORT	DB2AUDIOEXPORT	DB2VIDEOEXPORT	Used to resolve target file name for retrieve operations to a server file.

Table 17. Environment variables for DB2 extenders (continued)

Image Extender	Audio Extender	Video Extender	Description
DB2IMAGETEMP			Used to resolve target file name for operations that create temporary server files. However, if the TMP environment variable is specified, the directory TMP is used to resolve file names.
Client environment variables			
DB2IMAGEPATH	DB2AUDIOPATH	DB2VIDEOPATH	Used to resolve source file name for display and play operations on a client file.
DB2IMAGETEMP	DB2AUDIOTEMP	DB2VIDEOTEMP	Used to resolve target file name for operations that create temporary client files. However, if the TMP environment variable is specified, the directory TMP is used to resolve file names.

If you don't set the appropriate environment variable for the specific extender, the extender will use the following environment variables to resolve file names:

Environment variable	Description
DB2MMPATH	Used to resolve source file name for store, retrieve, and update operations.
DB2MMSTORE	Used to resolve target file name for store and update operations.
DB2MMEXPORT	Used to resolve target file name for retrieve operations.
DB2MMTEMP	Used to resolve file name for operations that create temporary files.

How environment variables are used to identify display or play programs

In addition to resolving file names, environment variables are also used to identify programs to display image objects retrieved by the Image Extender and play audio or video objects retrieved by the Audio and Video Extender. You use the DBiBrowse, DBaPlay, and DBvPlay APIs, respectively to display or play these objects. When you use each API, you can specify a display or play program or you can indicate that you want a default program to display or play the object.

The DB2 Extenders use the following environment variables in the client to identify the default display or play programs:

Environment variable	Description
DB2IMAGEBROWSER	Used to identify the default image display program.
DB2AUDIOPLAYER	Used to identify the default audio player program.
DB2VIDEOPLAYER	Used to identify the default video player program.

How the DB2MMDATAPATH environment variable is used (EEE only)

The DB2 extenders use the DB2MMDATAPATH environment variable to resolve locations for various operations in a partitioned database environment. For example, the DB2 Image Extender uses the value of DB2MMDATAPATH to store QBIC data in a partitioned database environment.

You set DB2MMDATAPATH when you create a DB2 extender instance, as described in any of the following the installation "readme" files:

- install.txt in the aixeee directory (Installing DB2 Extenders for use with DB2 Extended Enterprise Edition in AIX)
- install.txt in the soleee directory (Installing DB2 Extenders for use with DB2 Extended Enterprise Edition in Solaris Operating Environment)
- install.txt in the winnteee directory (Installing DB2 Extenders for use with DB2 Extended Enterprise Edition in Windows NT)

One example of how DB2MMDATAPATH is used is in storing QBIC feature and index data. In UNIX, the DB2 Image Extender stores this QBIC data in the following directory:

db2mmdatapath /NODE*node_num*/QBIC/*database_name*

where *db2mmdatapath* is the value of the DB2MMDATAPATH environment variable, *node_num* is the node number, and *database_name* is the database name.

Consider the following AIX example. Suppose DB2MMDATAPATH is set to /localfs/dmbdata. Suppose too that a database named sample is partitioned in nodes 0, 2, and 5. QBIC data will be stored for the sample database in the following directories:

Node 0: /localfs/dmbdata/NODE0000/QBIC/sample

Node 2: /localfs/dmbdata/NODE0002/QBIC/sample

Node 5: /localfs/dmbdata/NODE0005/QBIC/sample

Setting environment variables

You can set environment variables in AIX, HP-UX, Solaris, OS/2, and Windows.

Setting environment variables in AIX, HP-UX, and Solaris servers and clients

In AIX, HP-UX, and Solaris, the environment variables are specified in C shell, Korn shell, and Bourne shell scripts. When the DB2 extenders are installed, the environment variables for the server are set as follows:

C shell

```
setenv DB2MMPATH /usr/lpp/dmb_10/samples:/tmp
setenv DB2MMTEMP /tmp
setenv DB2MMSTORE /tmp
setenv DB2MMEXPORT /tmp
```

Korn and Bourne shell

```
DB2MMPATH=/usr/lpp/dmb_10/samples:/tmp
export DB2MMPATH

DB2MMSTORE=/tmp
export DB2MMSTORE

DB2MMEXPORT=/tmp
export DB2MMEXPORT

DB2MMTEMP=/tmp
export DB2MMTEMP
```

The environment variables for the server are initially set to values that allow you to access the media files used in the sample programs distributed with the DB2 Extenders. (See “Appendix B. Sample Programs and Media Files” on page 559 for information about the sample programs and media files.)

The client environment variables are set as follows when you install the DB2 Extenders in an AIX, HP-UX, or Solaris client:

C shell

```
setenv DB2MMPATH /tmp
setenv DB2MMTEMP /tmp
```

Korn and Bourne shell

```
DB2MMPATH=/tmp
export DB2MMPATH

DB2MMTEMP=/tmp
export DB2MMTEMP
```

Set the server and client environment variables that are used to resolve file names. Specify values that are appropriate for your environment. You can specify multiple directories, separated by a delimiter, for the environment variables that end in PATH. The environment variables that end in STORE, EXPORT, and TEMP are set with one directory only.

Specify the names of the appropriate image display, audio play, and video play programs in the DB2IMAGEBROWSER, DB2AUDIOPLAYER, and DB2VIDEOPLAYER client environment variables, respectively.

You can change the initial settings of the environment variables as follows:

C shell

Use the SETENV command to set environment variables:

```
setenv env-var directory
```

For example:

```
setenv DB2MMPATH /usr/lpp/dmb_10/samples:/media
setenv DB2IMAGEPATH /employee/pictures:/images
setenv DB2AUDIOSTORE /employee/sounds
setenv DB2IMAGEBROWSER 'xv %s'
```

Bourne shell

Use the EXPORT command to set environment variables:

```
env-var=directory
export env-var
```

For example:

```
DB2MMPATH=/usr/lpp/dmb_10/samples:/media
export DB2MMPATH

DB2IMAGEPATH=/employee/pictures:/images
export DB2IMAGEPATH

DB2AUDIOSTORE=/employee/sounds
export DB2AUDIOSTORE
```

Korn shell

Use the EXPORT command to set environment variables:

```
export env-var=directory
```

For example:

```
export DB2MMPATH=/usr/lpp/dmb_10/samples:/media
export DB2IMAGEPATH=/employee/pictures:/images
export DB2AUDIOSTORE=/employee/sounds
```

Setting environment variables in OS/2 servers and clients

In OS/2, the environment variables are added to your CONFIG.SYS file and automatically set during installation.

If you install the DB2 Extenders in an OS/2 server, the server environment variables are set as follows:

```
SET DB2MMPATH=install-dir\SAMPLES;temp-file-dir
SET DB2MMSTORE=temp-file-dir
SET DB2MMEXPORT=temp-file-dir
SET DB2MMTEMP=temp-file-dir
```

where *install-dir* is the installation directory and *temp-file-dir* is the temporary file directory. The default installation directory is C:\DMB, and the default temporary file directory is C:\DMB\TMP. You can change the location of either directory during installation. It is important that you correctly specify the location of the temporary file directory.

The environment variables for the server are initially set to values that allow you to access the media files used in the sample programs distributed with the DB2 Extenders. (See "Appendix B. Sample Programs and Media Files" on page 559 for information about the sample programs and media files.)

If you install the DB Extenders in an OS/2 client, the client environment variables are set as follows:

```
SET DB2MMPATH=temp-file-dir
SET DB2MMTEMP=temp-file-dir
```

Use the SET command to reset the environment variables. You can specify multiple directories, separated by a delimiter, for the environment variables that end in PATH. The environment variables that end in STORE, EXPORT, and TEMP are set with one directory only.

Use the SET command to specify the appropriate image display, audio player, and video player programs in the DB2IMAGEBROWSER, DB2AUDIOPLAYER, and DB2VIDEOPLAYER client environment variables, respectively.

Specify the SET command as follows:

```
SET env-var=directory
```

for example,

```
SET DB2MMPATH=C:\DMB\SAMPLES;\D:\MEDIA
SET DB2IMAGEPATH=C:\EMPLOYEE\PICTURES;D:\IMAGES
SET DB2AUDIOSTORE=C:\EMPLOYEE\SOUNDS
SET DB2IMAGEBROWSER=ib.exe %s
```

Setting environment variables in Windows servers and clients

In Windows, how you set environment variables depends on whether you are using DB2 Extenders in a non-partitioned environment or in a partitioned database environment (that is, with DB2 Extended Enterprise Edition for Windows NT).

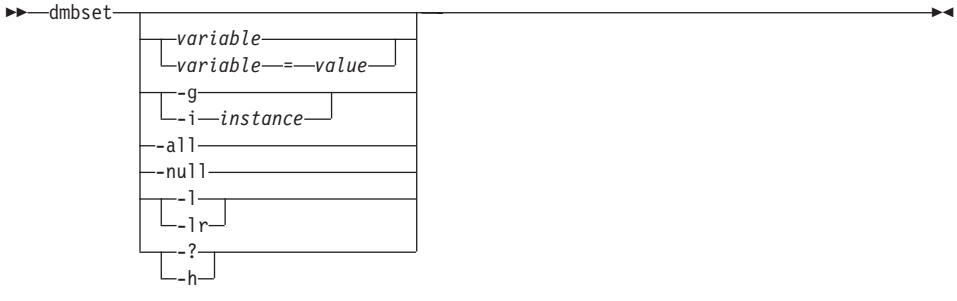
Setting environment variables in Windows non-partitioned database environments (Non-EEE Only)

In Windows NT, environment variables are stored in the system registry. Variables can be set by opening the Windows NT control panel and selecting the system icon. From the System Properties dialog, select the Environment tab. There are two windows containing environment variables and their values. The top window displays variables which are in effect for all users. The bottom window displays variables which are in effect for only the current user.

Setting environment variables in Windows NT partitioned database environments (EEE Only)

In a Windows NT partitioned environment, all variables used by DB2 extenders are stored in a private area of the system registry. A program called DMBSET is provided to inspect and modify extender variables.

The syntax of the program is:



To query the value of a variable, type dmbset *variable_name*. For example:

```
dmbset DB2MMPATH
```

Environment variables

To set the value of a variable, type: dmbset *variable_name=value*. For example:

```
dmbset DB2MMPATH=C:\DMB\SAMPLES
```

To display the value of all variables for a defined instance, type dmbset -i *instance_name*. For example:

```
dmbset -i dmbinst1
```

To set a value to null, type dmbset *variable_name -null*. For example:

```
dmbset DB2MMPATH -null
```

To display the value of the variables used by all instances, type dmbset -g.

To list the names of all variables used by DB2 extenders, type dmbset -lr.

To list the names of all instance profiles defined in the registry, type dmbset -l.

You have a lot of flexibility in setting environment variables for DB2 extenders in a partitioned database environment. For example, you can specify values for any environment variable, except DB2MMDATAPATH, in any of the following formats:

- Universal Naming Convention name: *machine_name**share_name*. For example:

  ```
  \\harmony\JimsShr
  ```

- Drive:path. For example:

  ```
  f:\media
  ```

- Anything else: *share_name**directory_name*. For example:

  ```
  JimsShr\images
  ```

Appendix B. Sample Programs and Media Files

Included with the DB2 extenders are various sample programs. The sample programs use image, audio, and video files that are also supplied with the extenders. All of the sample programs are in Call Level Interface (CLI) format.

The sample programs are installed in the SAMPLES subdirectory of the target directory when you install the DB2 extenders. The image, audio, and video files are also installed in the SAMPLES subdirectory of the target directory when you install the DB2 extenders. During installation, the extenders environment variables are set to point to the samples subdirectory of the target directory.

Sample programs

A number of files comprise the sample programs for the DB2 extenders. The files are:

File	Description
enable.c	Enables a database for the Audio, Image, and Video Extenders, creates a table, and enables the table and its columns.
populate.c	Imports data into the table
query.c	Queries the data in the table
api.c	Queries the database using extender APIs
handle.c	Demonstrates the use of handles in UDFs and how to make where clause comparisons in SELECT statements
qbcatdemo.c	Creates a QBIC catalog and catalogs a column of images into the catalog
qbicdemo.c	Queries a QBIC catalog
color.c	Makes color table declarations for qbicdemo·c
qbicpick.c	Presents color, histogram, and positional color selectors for a QBIC query.
shot.c	Creates a shot catalog for a video
makesf.c	Creates a shot catalog file for use with makehtml·exe.

Sample programs

makehtml.c	Accesses a shot catalog and creates HTML pages for display by a Web browser.
utility.c	Utility routines
utility.h	Header file for utility routines
makefile.aix	Makefile to build the programs in AIX
makefile.os2	Makefile to build the programs in OS/2
makefile.win	Makefile to build the programs in Windows 3.1
makefile.iva	Makefile to build the programs in Windows NT using IBM VisualAge C++
makefile.mvc	Makefile to build the programs in Windows 95 using Microsoft Visual C++
makefile.sun	Makefile to build the programs in Solaris
makefile.hp	Makefile to build the programs in HP-UX

Executable files are provided for the following sample programs. The sample programs are intended to be run in the order shown.

1. Enable
2. Populate
3. Query
4. API
5. Handle
6. Qbcatdmo
7. Qbicdemo
8. Qbicpick
9. Makesf
10. Makehtml

Prior to running the sample programs, you must create a database on your server. The extender services must have also been started on the server. To run a sample program, type the program name (this executes the program's executable file). You will be prompted for the database name, user ID, and password. Use the user ID and password of the user that created the database.

You can also build your own executable files for the sample programs. To do that, you need to:

1. Copy the sample program files to a writable directory.

2. Edit the makefile to specify the locations on your system where DB2, the extenders, and the compiler are installed.
3. Use make or nmake to compile the files into executable programs.

For further information about installing and using the sample programs, see the README.CNT file in the sample programs directory.

Sample image, audio, and video files

The sample image and audio files provided with the DB2 extenders are:

- Image Files
  - lizzi.bmp
  - sws_stri.bmp
  - nitecry.bmp
  - ranger_r.bmp
  - fuzzblue.bmp
- Audio Files
  - lizzi.wav
  - sws_stri.wav
  - nitecry.wav
  - ranger_r.wav
  - fuzzblue.wav
- Video Files
  - nitecry.avi
  - sample.mpg

Sample media files

Appendix C. Migrating to DB2 Extenders Version 5

If you use DB2 extenders Version 1, you need to migrate DB2 extender instances to DB2 extenders Version 5. This copies relevant information from Version 1 directories and files and merges this information with Version 5 information. You also need to migrate databases that contain administrative support tables. Migrating the databases to DB2 extenders Version 5 also migrates indexes used in QBIC operations.

Backup your database: Before you migrate to DB2 extenders Version 5, you should backup all of your DB2 databases. See *DB2 Universal Database Administration Guide* for information on backing up a DB2 database.

Migrating DB2 extender Version 1 instances

You can migrate a DB2 extender Version 1 instance in AIX, OS/2, or Windows NT. You first migrate the instance to DB2 Version 5. Then you migrate the instance to DB2 extenders Version 5.

In AIX

To migrate a DB2 extender instance in AIX:

1. Log in as sysadmin for the instance you are migrating.
2. Make sure that all databases you are migrating are cataloged.
3. End all applications that are currently using the instance.
4. Issue the `db2stop` and `db2 terminate` commands.
5. Issue the command `db2licd end`.

   This stops the DB2 license daemon.
6. Log in as root.
7. Issue the `db2imigr` command to migrate the existing instance to DB2 Version 5. The command checks for conditions that can prevent successful migration of the databases that are cataloged for the instance. If any adverse conditions exist, the command ends instance migration and generates a report that lists the detected conditions. The report is stored in a file called INSTHOME/migration.log.

   For details about using the `db2imigr` command, see *DB2 Universal Database for AIX Quick Beginnings*.
8. Start the db2ext command-line processor.
9. Issue the `dmbimigr` command as follows:

   `dmbimigr` *`instance_name`*

where *instance_name* is the name of the instance to be migrated. The `dmbimigr` command performs the following actions:

- Backs up the instance. The backup copy of the instance is moved from INSTHOME/dmb to INSTHOME/_v1.
- Creates a Version 5 instance. Relevant files and directories from the Version 1 instance are copied and migrated.
- Updates the list of instances.
- Copies the Version 1 dmbprofile to dmbprofile_v1 under the Version 5 instance.
- Copies the contents of the Version 1 directory to the Version 5 instance.

In OS/2 and Windows NT

Issue the `dmbimigr` command to migrate a DB2 extender instances in OS/2 or Windows NT. The format of the command is as follows:

```
dmbimigr oldpath newpath
```

where *oldpath* is the directory in which the old version of the DB2 extenders is installed, and *newpath* is the directory in which the new version of the DB2 extenders are installed. The `dmbimigr` command migrates all files and directories for all instances created under the previous version of the DB2 extenders.

Migrating DB2 databases

You can migrate DB2 databases in AIX, OS/2, or Windows NT to DB2 extenders Version 5. You first migrate your databases to DB2 Version 5. Then you migrate DB2 extender tables to DB2 extenders Version 5.

Schedule your database migration: Migrating DB2 databases can take a significant amount of time, especially if the databases contain many LOBs. You should schedule your database migration accordingly.

In AIX

To migrate DB2 databases in AIX:

1. Log in as the instance owner for the instance that contains the databases you are migrating.
2. Make sure that all the databases you are migrating are cataloged.
3. End all applications that are currently using the instance.
4. Issue the `db2stop` and `db2 terminate` commands.
5. Issue the command `db2licd end`.

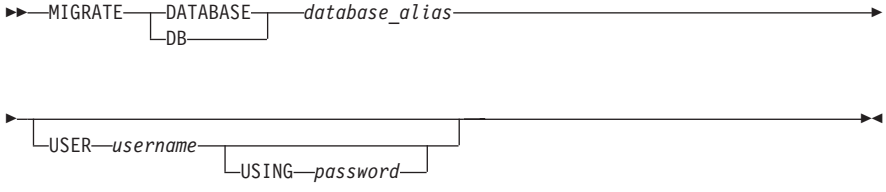
   This stops the DB2 license daemon.

6. Log in as root.
7. Issue the db2ckimg command from the /usr/lpp/db2_05_01 directory.

   DB2 uses the db2ckmig command to check for conditions that can prevent successful migration of the databases that are cataloged for the instance. If any adverse conditions exist, the command ends instance migration and generates a report that lists the detected conditions.

   For details about the db2ckimg command, see *DB2 Universal Database for AIX Quick Beginnings*.
8. Issue the migrate database command to migrate the database to DB2 Version 5, as described in *DB2 Universal Database Command Reference*.

   Alternatively, you can use the sqlemgdb API call in a program to migrate a database to DB2 Version 5. See *DB2 Universal Database API Reference* for a description of this API.
9. Migrate the DB2 extender tables in the database to DB2 extenders Version 5:
   - Start the db2ext command-line processor.
   - Log in as the instance owner.
   - Issue the following command:



   where:

   database_alias
   Specifies the alias name of the database.

   username
   Identifies the user name under which the database is being migrated.

   password
   The password used to authenticate the user name.

   Alternatively, you can issue the following API call in a program to migrate the DB2 extender tables in the database to DB2 extenders Version 5:

```
DMBMgDatabase(
      char *DbAlias,
      char *UserName
      char *Password
      );
```

where:

DbAlias
The alias name of the database.

UserName
The user name under which the database is being migrated.

Password
The password used to authenticate the user name.

When you migrate the DB2 extender tables in the database, you also migrate indexes used in QBIC operations.

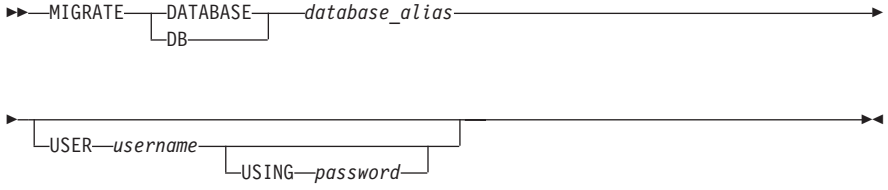
In OS/2 and Windows NT

To migrate DB2 databases in OS/2 or Windows NT:

1. Make sure that all the databases you are migrating are cataloged.
2. End all applications that are currently using the instance.
3. Issue the `db2stop` and `db2 terminate` commands.
4. Issue the `db2ckimg` command, as described in *DB2 Universal Database for OS/2 Quick Beginnings* or in *DB2 Universal Database for Windows NT and Windows 95 Quick Beginnings*.

   DB2 uses the `db2ckmig` command to check for conditions that can prevent successful migration of the databases that are cataloged for the instance. If any adverse conditions exist, the command ends instance migration and generates a report that lists the detected conditions.
5. Issue the `migrate database` command to migrate the database to DB2 Version, as described in *DB2 Universal Database Command Reference*.

   Alternatively, you can use the `sqlemgdb` API call in a program to migrate a database to DB2 Version 5. See *DB2 Universal Database API Reference* for a description of this API.
6. Migrate the DB2 extender tables in the database to DB2 extenders Version 5:
   - Start the db2ext command-line processor.
   - Enter the following command:



where:

database_alias
The alias name of the database.

username
The user name under which the database is being migrated.

password
The password used to authenticate the user name.

Alternatively, you can issue the following API call in a program to migrate the DB2 extender tables in the database to DB2 extenders Version 5:

```
DMBMgDatabase(
      char *DbAlias,
      char *UserName
      char *Password
      );
```

where:

DbAlias
The alias name of the database.

UserName
The user name under which the database is being migrated.

Password
The password used to authenticate the user name.

When you migrate the DB2 extender tables in the database, you also migrate indexes used in QBIC operations.

Appendix D. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make

improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This publication is intended to help you administer DB2 extenders and develop programs for use with DB2 extenders. This publication documents General-use Programming Interface and Associated Guidance Information provided by DB2 extenders .

General-use programming interfaces allow you to write programs that obtain the services of DB2 extenders. You may copy the DB2 extenders run-time feature needed for the application you develop onto client or server machines. To install the run-time feature, see the installation instructions provided in the README.TXT file for your operating system on the DB2 extenders CD-ROM.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX	DB2 Universal Database	PS/2
DB2	IBM	QBIC
DB2 extenders	OS/2	VisualAge

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company limited.

Intel is a registered trademark of Intel.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

administrative support tables. Tables used by a DB2 extender to process user requests on image, audio, and video objects. Some administrative support tables identify user tables and columns that are enabled for an extender. Other administrative support tables contain attribute information about objects in enabled columns. Also called a *metadata table*.

analyze. To calculate numeric values for the features of an image and add the values to a QBIC catalog.

application programming interface (API). An interface provided by a program that allows an application to use specific data or functions of the program. The DB2 extenders provide APIs for requesting user-defined functions, administrative operations, display operations, and video scene change detection.

audio. Pertaining to the portion of recorded information that can be heard.

average color. A measurement of color computed as an average of the color values contained in the pixels of an image.

audio clip. A section of recorded audio material.

binary large object (BLOB). A binary string whose length can be up to 2 GB. Image, audio, and video objects are stored in a DB2 database as BLOBs.

character large object (CLOB). A character string of single-byte characters, where the string can be up to 2 GB. CLOBs have an associated code page. Text objects that contain single-byte characters are stored in a DB2 database as CLOBs.

coarseness. An attribute of *texture* that measures the scale of the texture (pebbles versus boulders).

contrast. An attribute of *texture* that refers to the vividness of the pattern, and is a function of the variance of a grey-level histogram.

database partition. A part of a database. Each database partition is often (though not always) located on a separate machine.

database partition server. Manages a *database partition*. A database partition server is composed of a database manager and the collection of data and system resources that it manages. Typically, one database partition server is assigned to each machine. Also sometimes called a *node*.

directionality. An attribute of *texture* that describes whether the image has a favored direction (like grass) or is like a smooth object (like glass).

dissolve. To decrease the strength of a signal for a video frame as the strength of the signal for the next video frame increases.

double-byte character large object (DBCLOB). A character string of double-byte characters, or a combination of single-byte and double-byte characters, where the string can be up to 2 GB. DBCLOBs have an associated code page. Text objects that include double-byte characters are stored in a DB2 database as DBCLOBs.

feature. A visual attribute of an image, such as *average color*.

file reference variable. A programming variable that is useful for moving a LOB to and from a file on a client workstation.

gigabyte (GB). One billion (10^9) bytes. When referring to memory capacity, 1 073 741 824 bytes.

handle. A character string created by an extender that is used to represent an image, audio, or video object in a table. A handle is stored for an object in a user table and in *administrative support tables*. In this way, an

extender can link the handle stored in a user table with information about the object stored in the administrative support tables.

histogram color. A measurement of the distinct colors in an image. Data for each color is stored separately in a *QBIC catalog*.

host variable. A variable in an application program that can be referenced in embedded SQL statements. Host variables are the primary mechanism for transmitting data between a database and application program work areas.

image. An electronic representation of a picture.

index file. A file that contains indexing information used by the Video Extender to find a *shot* or an individual frame in a video clip.

kilobyte (KB). One thousand (10^3) bytes. When referring to memory capacity, 1024 bytes.

large object (LOB). A sequence of bytes, where the length can be up to 2 GB. A LOB can be of three types: *binary large object* (BLOB), *character large object (CLOB)* or *double-byte character large object (DBCLOB)*.

LOB locator. A small (4-byte) value stored in a host variable that can be used in a program to refer to a much larger LOB in a DB2 database. Using a LOB locator, a user can manipulate the LOB as if it was stored in a regular host variable, and without the need to transport the LOB between the application on the client machine and the database server.

megabyte (MB). One million (10^6) bytes. When referring to memory capacity, 1 048 576 bytes.

metadata table. *Administrative support table*.

multipartition nodegroup. A *nodegroup* that contains more than one *database partition server*.

node. *Database partition server*

nodegroup. A group of logically related *nodes*.

object. In object-oriented programming, an abstraction consisting of data and the operations associated with that data.

object orientation. A programming approach in which anything, real or abstract, can be represented in an application as an object that comprises a set of operations and data values. For example, a document can be represented by a document object that comprises document data and operations that can be performed on the document, such as filing, sending, and printing. A video clip can be represented by a video object that comprises video data and operations such as playing the video clip or finding a specific video frame.

partitioned database. A database that is composed of two or more *database partitions*.

pixel. The smallest element of an image that a screen can display.

positional color. The *average color* value of the pixels in a specified area of an image.

Query by Image Content (QBIC). A capability provided by the Image Extender that allows users to search images by their visual characteristics such as *average color* and *texture*.

QBIC catalog. A repository that holds data about the visual features of images.

query object. An object that specifies the features, feature, values, and feature weights for a QBIC query. The object can be named and saved for subsequent use in a QBIC query. Contrast with query string

query string. A character string that specifies the features, feature, values, and feature weights for a QBIC query. The query string can be entered in a query from the DB2 command line. Contrast with query object

scaling. Adding *nodes* to a database to increase storage space and performance.

scene change. A point in a *video clip* where there is a significant difference between two successive frames. This happens, for example, when a camera changes its point of view while recording a video.

score. A calculated value reflecting how similar the feature values are to those specified in a query by image content. The higher the number, the closer the match. The score is used to sort the results of a query by image content.

shot. The frames between two scene changes.

shot catalog. A database table or file that is used to store data about shots, such as the starting and ending frame number for a shot, in a video clip. A user can access a view of the table through an SQL query, or access the data in the file.

storyboard. A visual summary of a video. The Video Extender includes facilities to identify and store video frames that are representative of the shots in a video. These representative frames can be used in building a storyboard.

terabyte. A trillion (10^{12}) bytes. Ten to the twelfth power bytes. When referring to memory capacity, 1 099 511 627 776 bytes.

texture. One of the features that can be used in a query by image content. It refers to the coarseness, contrast, or directionality of an image.

thumbnail. A miniature image.

trigger. The definition of a set of actions to be taken when a table is changed. Triggers can be used to perform actions such as validating input data, automatically generating a value for a newly inserted row, reading from other tables for cross-referencing purposes, or writing to other tables for auditing purposes. Triggers are often used for integrity checking or to enforce business rules.

user-defined function (UDF). A function that is defined by a user to DB2. Once defined, the function can be used in SQL queries. and video objects. For example, UDFs can be created to get the compression format of a video or return the sampling rate of an audio. This provides a way of defining the behavior of objects of a particular type.

user-defined type (UDT). A data type defined by a user to DB2. UDTs are used to differentiate one LOB from another. For example, one UDT can be created for image objects and another for audio objects. Though stored as BLOBs, the image and audio objects are treated as types distinct from BLOBs and distinct from each other.

video. Pertaining to the portion of recorded information that can be seen.

video clip. A section of filmed or videotaped material.

video index. A file that the Video Extender uses to find a specific *shot* or frame in a video clip.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9107-01