

DB2 Universal Database for OS/390 and z/OS



Image, Audio, and Video Extenders Administration and Programming

Version 7

DB2 Universal Database for OS/390 and z/OS



Image, Audio, and Video Extenders Administration and Programming

Version 7

Note

Before using this information and the product it supports, please read the general information under “Appendix C. Notices” on page 421.

First Edition (March 2001)

This edition applies to Version 7 of DB2 Universal Database Server for OS/390 and z/OS, 5675-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not notes.

© Copyright International Business Machines Corporation 1998, 2001. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	vii
--------------------------	------------

Tables	ix
-------------------------	-----------

About this book	xi
----------------------------------	-----------

Who should use this book	xi
How to use this book	xi
Highlighting conventions.	xii
How to read the syntax diagrams	xii
Related information	xiii

Part 1. Introduction	1
---------------------------------------	----------

Chapter 1. Overview	3
--------------------------------------	----------

Exploiting DB2	3
Powerful new ways to search for information	3
The DB2 extenders	4
The SDK and run-time environments	4
Using the extenders	4
Examples	5
Example 1: Retrieving a video by its characteristics	5
Example 2: Searching for images by content	7
Operating environments	9

Chapter 2. DB2 extender concepts	11
---	-----------

Object-oriented concepts	11
Large objects	11
User-defined types	12
User-defined functions.	12
UDF and UDT names	13
Triggers	14
Extender data structures	14
Administrative support tables	14
Handles	15
QBIC catalogs	16

Chapter 3. How the extenders work	19
--	-----------

An extender scenario	19
Preparing a database server	20
Preparing a table	21
Altering a table	22
Inserting data into a table	24
Selecting data from a table	25
Displaying and playing objects	26
Updating data in a table	26
Deleting data from a table	27

Part 2. Administering image, audio, and video data	29
---	-----------

Chapter 4. Planning for DB2 extenders	31
--	-----------

Workload management considerations	31
The number of WLM environments	31

Performance objectives for WLM environments	32
Security considerations	32
Access to image, audio, and video objects in tables	32
Access to QBIC catalog tables	33
Access to content in files	33
EXECUTE authority	34
The MMDBSYS user ID	35
Authority to administer the extenders	35
Table space considerations	35
Backup and recovery considerations	35

Chapter 5. Administration overview	37
---	-----------

Administration tasks you can perform with the DB2 extenders	37
---	----

Chapter 6. Preparing data objects for extender data	41
--	-----------

Enabling database servers	41
Specifying table space	41
Specifying WLM environments	42
Specifying external security	42
Examples	42
Enabling tables	45
Enabling columns	48
Disabling data objects	49

Chapter 7. Tracking data objects and media files	51
---	-----------

Checking the status of data objects	51
Finding table entries that reference files	52
Finding files referenced by table entries	53
Checking if media files exist.	54

Chapter 8. Granting and revoking privileges on administrative support tables	55
---	-----------

Part 3. Programming for image, audio, and video data	57
---	-----------

Chapter 9. Programming overview	59
--	-----------

Using extender UDFs and APIs.	59
Tasks you can perform with extender UDFs and APIs.	60
Sample table for extender examples	60
Before you begin programming for DB2 extenders	61
Including extender definitions	63
Specifying UDF and UDT names	63
Transmitting large objects.	64
Handling return codes.	67
Preparing a DB2 extender application	67
Preparing a DB2 application.	67

Additional steps for DB2 extender applications	68
Unicode support	70

Chapter 10. Storing, retrieving, and updating objects 71

Image, audio, and video formats	71
Image conversion options	72
Storing an image, audio, or video object	73
DB2Image, DB2Audio, and DB2Video UDF formats	74
DB2ImageA, DB2AudioA, and DB2VideoA UDF formats	76
Storing an object that resides on the client	77
Storing an object that resides on the server	78
Specifying database or file storage	78
Identifying the format for storage	79
Storing an object with user-supplied attributes	81
Storing a thumbnail (image and video only)	82
Storing a comment	83
Retrieving an image, audio, or video object	84
Content UDF formats for retrieval	84
Retrieving an object to the client	85
Retrieving an object to a server file	87
Retrieving and using attributes	88
Retrieving comments	90
Updating an image, audio, or video object	90
Content UDF formats for updating	91
ContentA UDF formats for updating	92
Replace UDF formats for updating	93
ReplaceA UDF formats for updating	95
Updating an object from the client	96
Updating an object from the server	97
Specifying database or file storage for updates	97
Identifying the format for update	98
Updating an object with user-supplied attributes	99
Updating a thumbnail (image and video only)	100
Updating a comment	101

Chapter 11. Displaying or playing an image, audio, or video object 103

Using the display or play APIs	103
Identifying a display or play program	103
Specifying BLOB or file content	104
Specifying a wait indicator	105
Displaying a thumbnail-size image or video frame	105
Displaying a full-size image or video frame	106
Playing an audio or video	106

Chapter 12. Querying images by content 107

How to query by image content	107
Managing QBIC catalogs	108
Creating a QBIC catalog	108
Opening a QBIC catalog	110
Adding a feature to a QBIC catalog	111
Removing a feature from a QBIC catalog	112
Retrieving information about a QBIC catalog	112
Manually cataloging a column of images	114
Recataloging images	114
Closing a QBIC catalog	115

Deleting a QBIC catalog	115
QBIC catalog sample program	115
Building queries	122
Specifying a query string	123
Using a query object	125
Issuing queries by image content	131
Querying images	132
Retrieving an image score	133
QBIC query sample program	134

Part 4. Reference 143

Chapter 13. User-defined types (distinct types) and user-defined functions 145

Schema	145
User-defined types (distinct types)	145
User-defined functions	145
AlignValue	149
AspectRatio	150
BitsPerSample	151
BytesPerSec	152
Comment	153
CompressType	155
Content	156
ContentA	160
DB2Audio	162
DB2AudioA	164
DB2Image	167
DB2ImageA	170
DB2Video	172
DB2VideoA	174
Duration	176
Filename	177
FindInstrument	178
FindTrackName	179
Format	180
FrameRate	181
GetInstruments	182
GetTrackNames	183
Height	184
Importer	185
ImportTime	186
MaxBytesPerSec	187
NumAudioTracks	188
NumChannels	189
NumColors	190
NumFrames	191
NumVideoTracks	192
QbScoreFromName	193
QbScoreFromStr	195
QbScoreTBFromName	196
QbScoreTBFromStr	198
Replace	200
ReplaceA	203
SamplingRate	206
Size	207
Thumbnail	208
TicksPerQNote	210
TicksPerSec	211

Updater	212
UpdateTime	213
Width	214

Chapter 14. Application programming interfaces 215

DBaAdminGetInaccessibleFiles	216
DBaAdminGetReferencedFiles	218
DBaAdminIsFileReferenced	220
DBaDisableColumn	222
DBaDisableServer	224
DBaDisableTable	225
DBaEnableColumn	227
DBaEnableServer	229
DBaEnableTable	231
DBaGetError	233
DBaGetInaccessibleFiles	234
DBaGetReferencedFiles	236
DBaIsColumnEnabled	238
DBaIsFileReferenced	240
DBaIsServerEnabled	242
DBaIsTableEnabled	243
DBaPlay	245
DBaPrepareAttrs	247
DBiAdminGetInaccessibleFiles	248
DBiAdminGetReferencedFiles	250
DBiAdminIsFileReferenced	252
DBiBrowse	254
DBiDisableColumn	256
DBiDisableServer	258
DBiDisableTable	259
DBiEnableColumn	261
DBiEnableServer	263
DBiEnableTable	265
DBiGetError	267
DBiGetInaccessibleFiles	268
DBiGetReferencedFiles	270
DBiIsColumnEnabled	272
DBiIsFileReferenced	274
DBiIsServerEnabled	276
DBiIsTableEnabled	277
DBiPrepareAttrs	278
DBvAdminGetInaccessibleFiles	279
DBvAdminGetReferencedFiles	281
DBvAdminIsFileReferenced	283
DBvDisableColumn	285
DBvDisableServer	287
DBvDisableTable	288
DBvEnableColumn	290
DBvEnableServer	292
DBvEnableTable	294
DBvGetError	296
DBvGetInaccessibleFiles	297
DBvGetReferencedFiles	299
DBvIsColumnEnabled	301
DBvIsFileReferenced	303
DBvIsServerEnabled	305
DBvIsTableEnabled	306
DBvPlay	307
DBvPrepareAttrs	309
QbAddFeature	310

QbCatalogColumn	312
QbCloseCatalog	314
QbCreateCatalog	315
QbDeleteCatalog	317
QbGetCatalogInfo	319
QbListFeatures	320
QbOpenCatalog	322
QbQueryAddFeature	324
QbQueryCreate	326
QbQueryDelete	327
QbQueryGetFeatureCount	328
QbQueryGetString	329
QbQueryListFeatures	330
QbQueryNameCreate	332
QbQueryNameDelete	334
QbQueryNameSearch	335
QbQueryRemoveFeature	337
QbQuerySearch	339
QbQuerySetFeatureData	341
QbQuerySetFeatureWeight	343
QbQueryStringSearch	344
QbReCatalogColumn	346
QbRemoveFeature	348

Chapter 15. Administration commands for the client 351

Entering DB2 extender administration commands	351
Getting online help for DB2 extender commands	352
ADD QBIC FEATURE	353
CATALOG QBIC COLUMN	354
CLOSE QBIC CATALOG	355
CREATE QBIC CATALOG	356
DELETE QBIC CATALOG	358
DISABLE COLUMN	359
DISABLE SERVER	360
DISABLE TABLE	361
ENABLE COLUMN	362
ENABLE SERVER	363
ENABLE TABLE	365
GET EXTENDER STATUS	367
GET INACCESSIBLE FILES	368
GET QBIC CATALOG INFO	370
GET REFERENCED FILES	371
GRANT	373
OPEN QBIC CATALOG	375
QUIT	376
REMOVE QBIC FEATURE	377
REVOKE	378
TERMINATE	380

Chapter 16. Diagnostic information 381

Handling UDF return codes	381
Handling API return codes	382
SQLSTATE codes	382
Messages	386
Diagnostic tracing	404
Start tracing	404
Stop tracing	405
Reformat trace information	405

Part 5. Appendixes 407**Appendix A. Setting environment variables for DB2 extenders 409**

How environment variables are used to resolve file names	409
How environment variables are used to identify display or play programs	410
Setting environment variables	410
Setting environment variables in OS/390	410
Setting environment variables in AIX and Solaris clients	411
Setting environment variables in Windows clients	412

Appendix B. Sample programs and media files 413

Sample programs	413
Sample image, audio, and video files	420

Appendix C. Notices 421

Programming interface information	422
Trademarks	423

Glossary 425**Index 427**

Figures

1. A multimedia database table	5	16. Sample code that enables a column	49
2. A query that accesses videos	6	17. Sample code that checks if a database server is enabled	52
3. An application that accesses and plays videos	6	18. Sample code that checks if a file is referenced by user tables	53
4. Searching for images by content	7	19. Sample code that gets a list of referenced files	54
5. An application that searches for images by content	8	20. A table used in DB2 extender programming examples	61
6. Administrative support tables	15	21. An application that uses a DB2 extender	62
7. Handles.	16	22. Query by image content	107
8. The employee table	19	23. QBIC catalog sample program	117
9. The employee table with an audio column added	20	24. QBIC query sample program	136
10. Inserting data into a table.	24	25. Sample JCL	415
11. Selecting data from a table	25	26. Sample Bind	419
12. Displaying and playing objects	26	27. Setting up the CLI.INI file	420
13. Updating data in a table	27	28. Sample STEPLIB concatenation	420
14. Sample code that enables a database server	43		
15. Sample code that enables a table	47		

Tables

1. User-defined functions created by the Image Extender	21	8. QBIC Feature Names	111
2. User-defined functions created by the Audio Extender	23	9. Feature values that can be specified in query string	123
3. Administration tasks and facilities for the DB2 extenders	38	10. What the Image Extender examines in QbImageSource	127
4. Tasks you can perform with DB2 extender APIs	60	11. User-defined types created by the DB2 extenders	145
5. Formats that can be processed by the DB2 extenders	71	12. DB2 Extender UDFs	146
6. Image conversion options	72	13. SQLSTATE codes and associated message numbers	382
7. Attributes managed by the DB2 extenders	88	14. Environment variables for DB2 extenders	409

About this book

This book describes how to use DB2 extenders for OS/390 to prepare and maintain a DB2 UDB server for OS/390 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 for OS/390 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 UDB.

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 for OS/390. People who work with the Text Extender for OS/390 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 for OS/390. Read this part if you are new to administering or programming with the DB2 extenders for OS/390.

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

This part discusses planning considerations for DB2 extenders for OS/390. It also describes how to prepare and maintain a DB2 UDB for OS/390 database server for image, audio, and video data. Read this part if you need to plan for and administer a DB2 UDB for OS/390 database server 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 for OS/390 UDFs and APIs to request operations on image, audio, or video data. It also discusses considerations for building DB2 extender for OS/390 applications. 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 for OS/390 UDFs, APIs, administrative commands, and diagnostic information such as messages and codes. Read this part if you are familiar with DB2 extender for OS/390 concepts and tasks, but need information about a specific DB2 extender for OS/390 UDF, API, command, message, or code.

“Appendixes”

The appendixes describe:

- How to set environment variables that are used by the DB2 extenders for OS/390 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

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).

►—required item—►

- Optional items appear below the main path.

►—
| optional item |—►

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

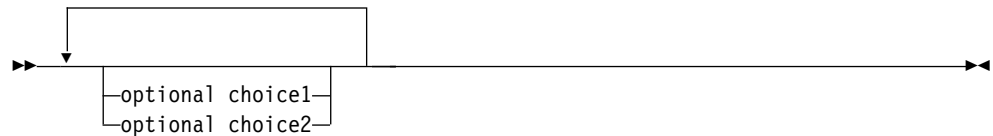
If you *must* choose one of the items, one item of the stack appears on the main path.



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 for OS/390 Version 7

DB2 Universal Database for OS/390 Version 7 Application Programming and SQL Guide, SC26-9004. This book describes how to design and write application programs that access DB2 UDB for OS/390.

DB2 Universal Database for OS/390 Version 7 SQL Reference, SC26-9014. This book describes the Structured Query Language (SQL) for DB2 UDB for OS/390.

DB2 Universal Database for OS/390 Version 6 ODBC Guide and Reference, SC26-9005. This book describes how to use DB2 CLI in applications to access DB2 servers.

DB2 Universal Database for OS/390 Version 6 Messages and Codes, GC26-9011. This book lists DB2 messages and codes, and specifies recovery actions.

DB2 Universal Database for OS/390 Version 6 Administration Guide, SC26-9003. This book describes how to administer DB2 for OS/390.

Program Directory for IBM Database 2 Universal Database Server for OS/390, Volume 1 of 8, Version 6, GI10-8182. This document includes installation and setup instructions for the DB2 extenders.

DB2 Text Extender for OS/390

DB2 Universal Database for OS/390 Version 6 Text Extender Administration and Programming, SC26-9651. This book describes how to administer a DB2 for OS/390 database server for text data. It also describes how to use

user-defined functions and application programming interfaces that are provided by the DB2 for OS/390 Text Extender to access and manipulate text data.

DB2 Universal Database Version 7. This is the workstation version of DB2.

DB2 Universal Database Application Development Guide, Version 6, SC09-2845.

Refer to this book for a description of how to prepare on a workstation client an application program that uses embedded SQL.

DB2 Universal Database CLI Guide and Reference, Version 6, SC09-2843. Refer to this book for a description of how to prepare on a workstation client an application program that uses CLI calls.

DB2 Universal Database Extenders Version 7. This is the workstation version of the DB2 extenders. It supports workstation clients and servers.

DB2 Universal Database Image, Audio, and Video Extenders Administration and Programming, Version 7 , SC26-9929. This book describes how to administer a DB2 UDB Version 7 database for image, audio, and video data. It also describes how to use application programming interfaces that are provided by the DB2 Image, Audio, and Video Extenders Version 6 to access and manipulate image, audio, and video data.

DB2 Universal Database Text Extender Administration and Programming, Version 7 , SC26-9930. This book describes how to administer a DB2 UDB Version 7 database for text data. It also describes how to use application programming interfaces that are provided by the DB2 Text Extender Version 6 to access and manipulate text data.

World Wide Web

DB2 extenders page. This page contains information about the DB2 extenders as well as technologies that are pertinent to the extenders. The URL of the DB2 extenders page is:

<http://www.ibm.com/software/data/db2/extendere>

Part 1. Introduction

Chapter 1. Overview	3
Exploiting DB2	3
Powerful new ways to search for information	3
The DB2 extenders	4
The SDK and run-time environments	4
Using the extenders	4
Examples	5
Example 1: Retrieving a video by its characteristics	5
Example 2: Searching for images by content	7
Operating environments	9
 Chapter 2. DB2 extender concepts	 11
Object-oriented concepts	11
Large objects	11
User-defined types	12
User-defined functions.	12
UDF and UDT names	13
Current path	13
Overloaded function names	14
Triggers	14
Extender data structures	14
Administrative support tables	14
Handles	15
QBIC catalogs	16
 Chapter 3. How the extenders work	 19
An extender scenario	19
Preparing a database server	20
Preparing a table	21
Altering a table	22
Inserting data into a table	24
Selecting data from a table	25
Displaying and playing objects	26
Updating data in a table	26
Deleting data from a table	27

Chapter 1. Overview

DB2 Universal Database Server for OS/390 (DB2 UDB for OS/390) 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 for OS/390, which are features of DB2 UDB for OS/390, help you exploit DB2 UDB for OS/390's object-relational features. (In this book, "DB2" refers to DB2 UDB for OS/390, and either "DB2 extenders" or simply "extenders" refers to DB2 extenders for OS/390.) 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.

Exploiting DB2

The DB2 extenders exploit the object-oriented features of DB2. In particular, with DB2 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 requested 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.

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

New ways to search

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*. For information about the XML Extender, see *XML Extender Administration and Programming*.

The SDK and run-time environments

The DB2 extenders installation package provides a Software Developers Kit (SDK) and client and server run-time environments. You can develop DB2 extender applications on a client or server machine in which you have installed the DB2 extender SDK.

You can run DB2 extender applications in a server machine that includes the DB2 extender client run-time code and server run-time code. (The client run-time code is automatically installed when you install the server run-time code.) You can also run DB2 extender applications on a client machine in which the DB2 extender client run-time code is installed. If you run an extender application from a client machine, you need to ensure that a connection can be made to the server.

Using the extenders

You can request the extender UDFs in a DB2 application program, or you can request them interactively using tools such as SPUFI from an OS/390 client or using the DB2 command-line processor from a workstation client.

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

- Administrative APIs to prepare and maintain a database server 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.)

The DB2 extenders also provide a command-line processor, referred to as the db2ext command-line processor, that you use to issue administrative commands. These commands access administrative APIs. You can start the db2ext command-line processor from a workstation client, or start it from an OS/390 client using OS/390 Open Edition services.

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 UDB and the DB2 extenders, the agency now also stores the content of the advertisements in the database. This includes images of print advertisements, videos of television advertisements, and recordings of radio advertisements. As Figure 1 shows, all of the related advertising information is in one database table that is named ADS.



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 advertisements created for the IBM account in 1997, but only advertisements whose duration is 30 seconds or less.

Figure 2 on page 6 shows a query that accesses the videos. Notice that the Video Extender UDFs named Filename and Duration in the query.

Examples

```
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 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.

```
#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;
```

Figure 3. An application that accesses and plays videos (Part 1 of 2)

```
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 (Part 2 of 2)

Example 2: Searching for images by content

A graphic illustrator in the advertising agency is developing a new print advertisement for a client. The illustrator wants to use a particular shade of blue in the background of the advertisement, 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 5). 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 8 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 Version 7 operate with DB2 Universal Database Version 7.1 (or higher) in a client/server environment. The minimum version and release levels that are required for the supported platforms are the same as those for DB2 Universal Database Version 7.1.

The supported client platforms are: OS/390, AIX, Windows NT[®] and later, Windows 98, Windows 95, and Solaris Operating Environment.

The supported server is OS/390.

Another DB2 extenders product, DB2 Universal Database Extenders Version 7, supports other server platforms, such as OS/2, AIX, Windows NT and later, Solaris Operating Environment, and HP-UX. For information about DB2 Universal Database Extenders Version 7, see *DB2 Universal Database Image, Audio, and Video Extenders Administration and Programming, Version 7* or *DB2 Universal Database Text Extender Administration and Programming, Version 7*.

The minimum version and release levels that are required for the supported platforms are the same as those for DB2 for OS/390 Version 7. The Language Environment and the compiler should be at the OS/390 Version 2 Release 4 level. For further information about minimum requirements see *Program Directory for IBM Database 2 Universal Server for OS/390 Version 7, Volume 1 of 8*. Workstation client support prerequires installation of the appropriate DB2 Client Application Enabler for the client workstation platform, and installation of the DB2 Connect feature. For further information about prerequisites for DB2 Extender workstation clients, see the installation README file in the DB2 Universal Database for OS/390 Version 7 Extenders Clients CD-ROM.

Examples

Chapter 2. DB2 extender concepts

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

Topic	See
Object-oriented concepts	Page 11
Extender data structures	Page 14

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

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

Object-oriented concepts

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.

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. The file must be in a file system that is compatible with OS/390 UNIX services, for example, a hierarchical file system (HFS).

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 'SCALEMAP'
  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'
```

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 current path to determine the function or distinct data type that you want.

Current path

The **current 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 current path by specifying the SQL statement `SET CURRENT PATH`. This sets the current path in the `CURRENT PATH` special register.

For the DB2 extenders, it is a good idea to add the `mmdbsys` schema to the current 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 current path:

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

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

Object-oriented concepts

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.parno));
  END
```

The DB2 extenders create and maintain administrative support tables to record information about image, audio, and video data stored in a database server. (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 server.

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.

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”.

As Figure 6 on page 15 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.

A special set of administrative support tables for image objects holds data about visual features such as average color and texture. These tables comprise a QBIC catalog. See “QBIC catalogs” on page 16 for more information.

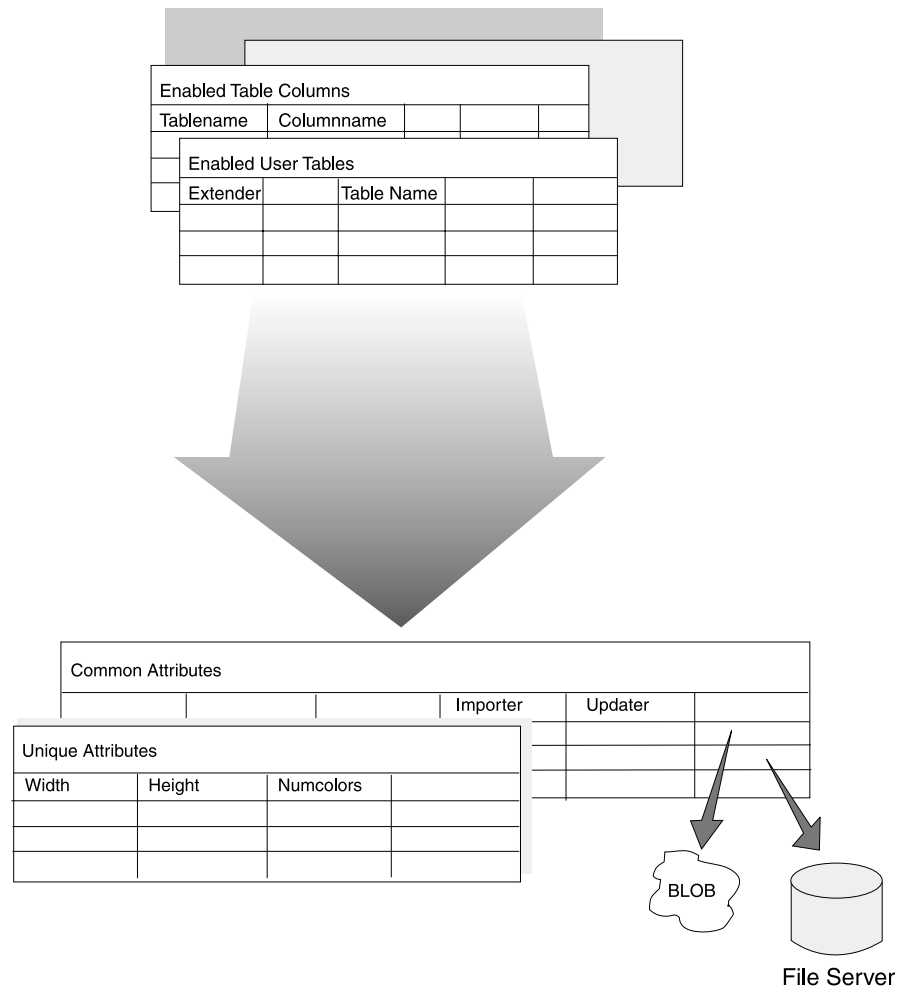


Figure 6. Administrative support tables

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

Data structures

information stored in the administrative support tables. Figure 7 illustrates the information stored for two images in a user table.

User Table

ID	Name	Picture
		Handle 1
		Handle 2

Administrative Support Tables

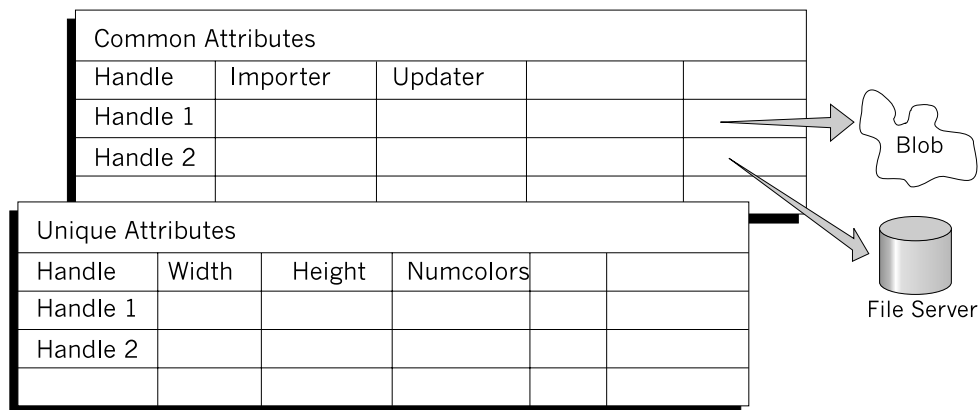


Figure 7. Handles

QBIC catalogs

A **QBIC catalog** is a set of administrative support tables that holds 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. The Image Extender records in an administrative support table the association between user table columns and QBIC catalogs. 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 one or more features for the search (such as average color), a source for each 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.

Chapter 3. How the extenders work

The DB2 extenders do a lot of work to handle image, audio, and video data requests. 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) that includes pictures of each employee.

A Database with pictures: As Figure 8 shows, an employee table in the database will contain the identification and name of each employee, as well as the employee's picture.

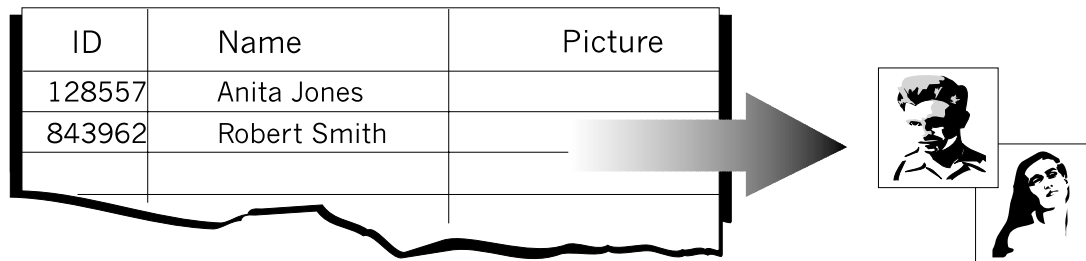


Figure 8. The employee table

To prepare the personnel database for image processing, a system administrator, that is, someone with SYSADM authority, 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 audio recording for each employee to the table. This is shown in Figure 9 on page 20.

Scenario

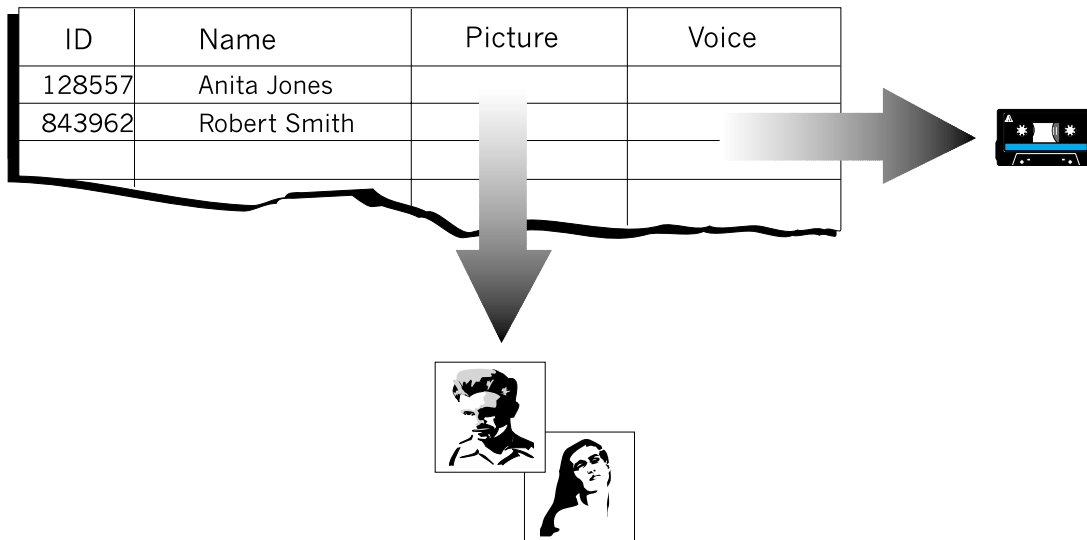


Figure 9. The employee table with an audio column added

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.

Preparing a database server

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

What the system administrator does: The system administrator creates the personnel database in the locally-attached database server using the following SQL statement:

```
CREATE DATABASE personnl;          /*create database*/
```

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

```
ENABLE SERVER FOR DB2IMAGE WLM ENVIRONMENT DMBWLM1
```

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

- Creates a user-defined type that is named DB2IMAGE for image objects.
- Creates administrative support tables for image objects.
- Creates user-defined functions for image objects. The UDFs will run in the MVS Workload Manager (WLM) environment that is named DMBWLM1. Because no table space specification was made in the ENABLE SERVER command, table spaces in the default storage group for the database server will be used to hold administrative support tables and their indexes. Because no external security specification was made in the ENABLE SERVER command, the default, EXTERNAL SECURITY DB2, is assumed.

The UDFs are listed in Table 1 on page 21.

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
ContentA	Update the content of an image with user-supplied attributes
DB2Image	Store the content of an image
DB2ImageA	Store the content of an image with user-supplied attributes
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
Replace	Update the content and user comments for an image
ReplaceA	Update the content and user comments for an image with user-supplied attributes
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 path by using the following SQL statement:

```
SET CURRENT PATH = mmdbsys, CURRENT 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 13 for more information about UDT and UDF names.

The DBA creates the employee table by issuing 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 USING TBSPACE1,,LTBSPACE1
ENABLE COLUMN employee picture FOR DB2IMAGE
```

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

Preparing a table

- Identifies the employee table for use.
- Creates administrative support tables that hold attribute information for image objects in enabled columns. The administrative support tables are stored in a table space that is named TBSPACE1.
- Creates an auxiliary LOB table to hold LOB data for enabled columns. The LOB table is stored in a LOB table space that is named LTBSpace1.
- Creates indexes for the administrative support tables and the auxiliary LOB table.
- Default table spaces will be used for indexes on the administrative support tables table space and the LOB data table space.

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 administrator does: The system administrator uses the db2ext command-line processor to enable the database server for use by the Audio Extender:

```
ENABLE SERVER FOR DB2AUDIO WLM ENVIRONMENT DMBWLM1
```

Notice that the DBA specifies the same command parameters to enable the server for use by the Audio Extender as for the Image Extender. In general, whatever parameters are specified on the ENABLE SERVER command for one extender should be specified for the other extenders.

The DBA then issues the following SQL statement to alter the employee table. The DBA uses the DB2 command line processor to issue the 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 USING TBSPACE1,,LTBSpace1  
ENABLE COLUMN employee voice FOR DB2AUDIO
```

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

- Creates a user-defined type that is named DB2AUDIO for audio objects.
- Creates administrative support tables for audio objects.
- Creates user-defined functions for audio objects. The UDFs will run in the WLM environment that is named DMBWLM1. DB2's default table space will be used to hold the administrative support tables and their indexes. EXTERNAL SECURITY DB2 is assumed.

The UDFs are listed in Table 2 on page 23.

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
ContentA	Update the content of an audio with user-supplied attributes
DB2Audio	Store the content of an audio
DB2AudioA	Store the content of an audio with user-supplied attributes
Duration	Get the playing time of an audio
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
ReplaceA	Update the content and user comments for an audio recording with user-supplied attributes
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. The administrative support tables are stored in table space TBSPACE1.
- Creates an auxiliary LOB table to hold LOB data for enabled columns. The LOB table is stored in the LOB table space that is named LTSPACE1.
- Creates indexes for the administrative support tables and the auxiliary LOB table.

Altering a table

- Default table spaces will be used for indexes on the administrative support tables table space and the LOB data table space.

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 (the table entry refers to the server file).

What the user does: The user inserts the record into the employee table by using an application program that includes the statements that are shown in Figure 10.

```
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 10. 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 that is 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; 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 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 that is named `ajones.wav`; the administrative support table record refers to the file.

- Stores a record in another administrative support table. The record contains audio-specific attributes such as the sampling rate of the audio.

Triggers 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 by using an application program that includes the SQL statements that are shown in Figure 11.

```
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 11. 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 on a workstation client (multimedia players are normally run on workstation clients). 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 by using an application program that includes the SQL statements that are shown in Figure 12.

```
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 12. 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 by using an application program that includes the SQL statements that are shown in Figure 13.

```
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 13. 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. The Image Extender uses the attributes of the new image to update the attributes stored in the administrative support tables for the old image. The image source is in a server file that is 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.

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 by 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.

Deleting data

Part 2. Administering image, audio, and video data

Chapter 4. Planning for DB2 extenders 31

Workload management considerations 31

 The number of WLM environments 31

 Performance objectives for WLM environments 32

Security considerations 32

 Access to image, audio, and video objects in
 tables 32

 Access to QBIC catalog tables 33

 Access to content in files 33

 EXECUTE authority 34

 The MMDBSYS user ID 35

 Authority to administer the extenders 35

Table space considerations 35

Backup and recovery considerations 35

Chapter 5. Administration overview 37

Administration tasks you can perform with the DB2
extenders 37

Chapter 6. Preparing data objects for extender data 41

Enabling database servers 41

 Specifying table space 41

 Specifying WLM environments 42

 Specifying external security 42

 Examples 42

Enabling tables 45

Enabling columns 48

Disabling data objects 49

Chapter 7. Tracking data objects and media files 51

Checking the status of data objects 51

Finding table entries that reference files 52

Finding files referenced by table entries 53

Checking if media files exist. 54

Chapter 8. Granting and revoking privileges on administrative support tables. 55

Chapter 4. Planning for DB2 extenders

Before you use the DB2 extenders, you need to make some basic decisions about how they will be run. In particular, you need to make decisions about:

- Workload management
- Security
- Table space
- Backup and recovery

This chapter describes considerations for making these decisions.

Workload management considerations

A WLM Application Environment is a set of parameters describing how to create address spaces which can run a particular kind of work. The extenders use WLM Application Environments for user-defined functions and for stored procedures. The extenders use stored procedures to process API requests.

After the DB2 extenders are installed, you need to establish WLM environments for the extender UDFs and stored procedures. (See *Program Directory for IBM Database 2 Universal Database Server for OS/390 Version 6, Volume 1 of 8* for instructions on how to install the extenders.) Each WLM environment is associated with a JCL procedure that starts an address space for executing the DB2 extender UDFs.

You need to decide how many WLM environments to establish. You also need to decide what performance objectives to specify for these environments.

The number of WLM environments

You can establish multiple WLM environments for running DB2 extender UDFs. When you enable a database server for a DB2 extender, you specify the WLM environment names (see “Specifying WLM environments” on page 42). You are allowed to specify up to two WLM environment names. If you specify one WLM environment name, then all of the extender’s UDFs run in that WLM environment.

If you specify two WLM environment names, the second WLM environment is used to run:

- UDFs that store, retrieve, or update objects (such as the DB2Image, Content, and Replace UDFs)
- Stored procedures for QBIC APIs

The first WLM environment is used for:

- Attribute retrieval UDFs (such as the Width, Height, and Size UDFs)
- UDFs that require longer processing times or that have higher expected memory requirements
- Stored procedures for administrative APIs

Specify one WLM environment: Unless you expect high DB2 extender workloads, you should specify one WLM environment when you enable a database server. In high workload situations, running extender UDFs in multiple WLM environments

Workload management considerations

can improve performance. However there is extra overhead in maintaining the multiple address spaces associated with two WLM environments.

Performance objectives for WLM environments

WLM can operate in either of two modes: compatibility mode or goal mode. In compatibility mode, work requests are given a service class by the classification rules in the active WLM service policy.

In goal mode, work requests are also assigned a service class by the classification rules in the active WLM service policy. However each service class period has a performance objective, that is, a goal. WLM raises or lowers that period's access to system resources as needed to meet the specified goal. For example, the goal might be "application APPL8 should run in less than 3 seconds of elapsed time 90% of the time".

Specify goal mode: In goal mode, WLM automatically starts WLM-established address spaces for user-defined functions to help meet the service class goals that you set. By comparison, in compatibility mode, WLM cannot automatically start a new address space to handle high-priority requests. Instead, you must monitor the performance of UDFs to determine how many WLM-managed address spaces to start manually. As a result, goal mode is recommended for running DB2 extender UDFs.

Security considerations

Before you use the DB2 extenders, you need to consider the implications the extenders have on security. For example, you need to determine what controls (if any) to put in place for access to image, audio, and video object content and metadata. You also need to determine whether you want to restrict privileges that the DB2 extenders automatically grant to users.

Access to image, audio, and video objects in tables

Image, audio, and video objects stored as BLOBs in a DB2 database are afforded the same security protection as traditional numeric and character data. Users must have the required privilege to select objects from, insert objects into, update objects in, or delete objects from a DB2 database. For example, to select objects from a user table, a user must have SELECT privilege on the table. For information about DB2 security, see the *DB2 Administration Guide*.

Users issue UDFs to select, insert, update, or delete objects from a user table. To perform the requested operations, the UDFs must be able to access, and if necessary update, the administrative support tables that hold attribute information for the objects. For the owner of a user table, the extenders automatically give the UDFs the access they need to handle the requested operation.

However users other than the table owner who need to select an object from the user table must be granted SELECT privilege on the administrative support tables. The extenders provide a GRANT command to grant privileges on the administrative support tables. The extenders also provide a REVOKE command to revoke privileges granted on administrative support tables. For further information about granting and revoking privileges on administrative support tables, see "Chapter 8. Granting and revoking privileges on administrative support tables" on page 55.

For insert, update, or delete operations, the extenders check to determine if the user has the needed INSERT, UPDATE, or DELETE privilege on the user table. If the user has the required privilege, the extenders allow the UDFs to access the administrative support tables, as required.

Access to QBIC catalog tables

Users who perform QBIC operations on image objects require appropriate privileges on the administrative support tables that comprise the QBIC catalog for those objects. For example, a user who issues a QBIC query against a column of images must have SELECT privilege on the QBIC catalog tables for the image column. A user who makes changes to the QBIC catalog should have SELECT, INSERT, UPDATE, and DELETE privilege on the associated QBIC catalog tables.

Other than the owner of the QBIC catalog (that is, the owner of the user table for which the catalog is specified), a user must be granted the appropriate privileges on the QBIC catalog tables. The extenders provide a GRANT command to grant privileges on the QBIC catalog tables. The extenders also provide a REVOKE command to revoke privileges on the QBIC catalog tables. For further information about granting and revoking privileges on QBIC catalog tables, see “Chapter 8. Granting and revoking privileges on administrative support tables” on page 55.

Grant privileges on a QBIC catalog after all features are added: Privileges granted on a QBIC catalog include privileges on QBIC feature tables, but only for features that have been already been added to the catalog. If you add a feature to the catalog after you grant privileges on the catalog, you will have to grant privileges on the catalog again. So you should grant privileges on a QBIC catalog only after the catalog is created and after all the features have been added.

Access to content in files

The image, audio, and video objects that you store in a table can point to content stored in files. The files must be in a file system that is compatible with OS/390 UNIX services, for example, a hierarchical file system (HFS).

When an administrator enables a database server for an extender, the administrator can specify an EXTERNAL SECURITY option (see “Specifying external security” on page 42). The option indicates how UDFs that store, retrieve, and update objects interact with an external security product such as RACF to control access to files. The administrator can specify EXTERNAL SECURITY USER or EXTERNAL SECURITY DB2.

If EXTERNAL SECURITY USER is specified, the DB2 extender UDFs run with the primary authorization ID of the process that called them. In addition, the UDFs also have permissions as defined for them on the DB2 server. The primary authorization ID of the process is used rather than other DB2 authorization IDs, such as the authorization ID of the package or plan owner. The primary authorization ID is subject to distributed database security operations such as inbound authorization ID translation.

If EXTERNAL SECURITY DB2 is specified, the DB2 extender UDFs access files using the authorization ID associated with the WLM environment address spaces that are established for running the UDFs. In this case, all extender users have access to the same files.

When a UDF attempts to access a file, OS/390 Open Edition Services calls an external security product such as Security Server (RACF) to get the user ID (UID)

Security considerations

and group ID (GID) associated with the UDF. For EXTERNAL SECURITY USER, the UID and GID are those that are assigned to the authorization ID in effect for the process that calls the UDF. For EXTERNAL SECURITY DB2, the UID and GID are those that are assigned to the authorization ID of the WLM application environment address spaces for the UDF. The system then compares the UID and GID assignments to the user, group, and other permission bits in the file's directory entry. The file can be accessed only if the user's UID and GID are compatible with the permissions in the file's directory entry.

EXTERNAL SECURITY USER gives greater control over file access: If you specify EXTERNAL SECURITY USER, filesystem checks are made against the primary authorization ID of the process that calls the UDF. Because you can assign different UIDs and GIDs to different users, you can control access to files on a user-by-user basis. By comparison, EXTERNAL SECURITY DB2 gives you one level of control because all UDFs run with the same UID and GID, that is, the UID and GID assigned to the WLM environment address spaces. For this reason, EXTERNAL SECURITY DB2 is a good choice for applications where file read protection is not required, for example Web applications.

EXTERNAL SECURITY DB2 requires less administration: If you specify EXTERNAL SECURITY DB2, you need to assign an authorization ID, UID, and GID to the WLM address spaces for the extender UDFs. By comparison, if you specify EXTERNAL SECURITY USER, you must assign a UID and GID for every legitimate user of the files. In both cases, you need to coordinate the UID and GID assignments with the filesystem permissions.

EXTERNAL SECURITY DB2 results in better UDF performance: This is because the individual nature of performing the security checks for EXTERNAL SECURITY USER incurs more overhead in the database server than EXTERNAL SECURITY DB2.

EXECUTE authority

When a database server is enabled for a DB2 extender, use privilege on the extender's UDT (and related CAST functions) and use privilege on all of its UDFs are granted to PUBLIC. You can revoke the use privilege on the UDT and UDFs that was granted to PUBLIC, and grant the privilege to use the UDT and UDFs to specific authorization IDs. This does not affect the way the extender operates. However maintaining authority lists could become tedious. Because of this, consider controlling access to files that are used (or potentially used) by DB2 extender UDFs, as described in "Access to content in files" on page 33. In effect, this limits the ability to successfully retrieve objects of the associated user-defined type to specific authorization IDs.

This has implications for external security: If you specify EXTERNAL SECURITY DB2, UDF access to files is controlled by authorization ID, UID, and GID specifications made for the WLM environment address spaces in which the UDFs run. However because EXECUTE authority on DB2 extender UDFs is automatically granted to PUBLIC, it means that anyone with INSERT or UPDATE privilege on an enabled table might have significant access to the HFS file system. So assign the authorization ID, UID, GID and file system permissions for these address spaces with care. It is important to restrict the file access for these address spaces to the minimum level required, based on your needs.

The MMDBSYS user ID

The DB2 extenders use an SQL ID of MMDBSYS. As a result, you should create an MMDBSYS user ID to manage DB2 extender objects such as administrative support tables. Use an appropriate external security system such as RACF to create the MMDBSYS user ID. If secondary authorization IDs are used by DB2, you should take steps to secure MMDBSYS as a secondary ID.

Authority to administer the extenders

Some extender-related administrative operations require special authority. See “Chapter 14. Application programming interfaces” on page 215 for the authority required by DB2 extender administrative APIs. See “Chapter 15. Administration commands for the client” on page 351 for the authority required by DB2 extender administrative commands.

Table space considerations

The DB2 extenders store attribute data and LOBs in administrative support tables that are contained in DB2 table spaces. When you enable a database server for an extender, you can specify a table space for the “global” administrative support tables for the database server (see “Specifying table space” on page 41). These administrative support tables store information such as the names of extenders for which the database server is enabled. The table space that you specify for the global administrative support tables, should be in the MMDBSYS database. The MMDBSYS database is created as part of the setup done after the DB2 extenders are installed. (See *IBM Database 2 Universal Database Server for OS/390 Version 6, Volume 1 of 8* for further information.)

When you enable a table for an extender, you must specify a table space for the attribute data, and a table space for LOBs stored in extender-enabled columns. The table space should be a segmented table space. Specify LOCKSIZE ROW when you create the table space if you expect any of the following actions to occur frequently, occur in complex transactions, or occur in units of work that are not immediately committed:

- Enable or disable operations
- Requests to create a QBIC catalog
- Requests to add a feature to a QBIC catalog

Specify a table space for the global administrative support tables: If you do not specify a table space, DB2 creates a table space for each global administrative support table. It is probably more efficient to specify a single, segmented table space for the global administrative support tables.

Specify table spaces in the same database as the user table: The table spaces that you specify when you enable a table for an extender should be in the same database as the table. One advantage of doing this is that objects and their metadata can be managed together.

Backup and recovery considerations

You need to back up the MMDBSYS database. The database is created as part of DB2 extenders initialization. (For further information about DB2 extenders initialization see *Program Directory for IBM Database 2 Universal Database Server for OS/390 Volume 1 of 8*.) The database contains the global metadata tables that keep track of which extenders are enabled on the database server, and which tables and

Backup and recovery considerations

columns are enabled for the extenders. The database also contains a list of the QBIC catalogs. You should back up the database after significant events occur related to enablement and QBIC catalogs. For example, back up the database after you enable a database server, create a QBIC catalog, or add a feature to a QBIC catalog.

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.

Chapter 5. Administration overview

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

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

- Administration application programming interfaces (APIs). You can include the DB2 extender APIs in your C language program. See “Chapter 14. Application programming interfaces” on page 215 for reference information on these APIs.
- Administration commands. You can submit administration commands to the db2ext command-line processor. See “Chapter 15. Administration commands for the client” on page 351 for instructions on entering administration commands and for additional reference information.

Administration tasks you can perform with the DB2 extenders

There are three categories of administrative tasks:

- Preparing data objects for extender data. You prepare database servers, 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.
- Tracking data objects and media files. As you debug applications that use the DB2 extenders, it is useful to know which data objects are enabled for extender data. It is also useful to understand the correlation between user tables and external media files.
- Granting and revoking authority on administrative support tables. To select an object from a table, a user needs to be granted SELECT privilege on the administrative support tables that hold attribute information for the object. If access to objects in a table is no longer appropriate for a specific user, you can revoke the user’s SELECT privilege and other privileges that the user has on the administrative support tables.

Table 3 on page 38 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, x represents the third character of each API statement. 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 DBiEnableTable, the API for enabling a table for audio is DBaEnableTable, and the API for enabling a table for video is DBvEnableTable. 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.

Administration overview

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 12. Querying images by content" on page 107.

Table 3. Administration tasks and facilities for the DB2 extenders

Task	Extender API	Extender Command	See
Preparing data objects for multimedia data			
Enable a database server	DBxEnableServer	ENABLE SERVER	p. 41
Disable a database server	DBxDisableServer	DISABLE SERVER	p. 49
Enable a table	DBxEnableTable	ENABLE TABLE	p. 45
Disable a table	DBxDisableTable	DISABLE TABLE	p. 49
Enable a column	DBxEnableColumn	ENABLE COLUMN	p. 48
Disable a column	DBxDisableColumn	DISABLE COLUMN	p. 49
Tracking data objects and media files			
Find out if database servers are enabled	DBxIsServerEnabled	GET EXTENDER STATUS	p. 51
Find out if tables are enabled	DBxIsTableEnabled	GET EXTENDER STATUS	p. 51
Find out if columns are enabled	DBxIsColumnEnabled	GET EXTENDER STATUS	p. 51
Find table entries that reference files in tables whose qualifier is the current user ID	DBxIsFileReferenced	No	p. 52
Find table entries that reference files in all tables of a specific qualifier or all tables in a database	DBxAdminIsFileReferenced	No	p. 52
Find files referenced by table entries in tables whose qualifier is the current user ID	DBxGetReferencedFiles	GET REFERENCED FILES	p. 53
Find files referenced by table entries in all tables of a specific qualifier or all tables in a database	DBxAdminGetReferencedFiles	GET REFERENCED FILES	p. 53
Find inaccessible files referenced by table entries in all tables whose qualifier is the current user ID	DBxGetInaccessibleFiles	GET INACCESSIBLE FILES	p. 54
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. 54
Granting and revoking privileges on administrative support (metadata) tables			
Grant privileges on administrative support tables	No	GRANT	p. 55
Revoke privileges on administrative support tables	No	REVOKE	p. 55

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

use DB2 commands or statements to perform some tasks. You perform other tasks with the DB2 extenders. This sequence assumes that your DB2 system is running.

Required tasks:

1. Connect to the database server.
2. Enable the database server.
3. Create a table and column (by using DB2).
4. Enable a table in the database.
5. Enable a column in the table.

Optional tasks:

1. Track data objects and media files.
2. Set the current path (using DB2).
3. Grant or revoke privileges on administrative support tables

Examples: Most of the examples in the next three chapters assume that 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 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 13.

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

Administration overview

Chapter 6. Preparing data objects for extender data

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

Use the DBxEnableServer API (where x is a for audio, i for image, or v for video) or the ENABLE SERVER command to enable a database server for a DB2 extender.

When you enable a database server, 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 that type.
- Creates administrative support tables (also called metadata tables) for the database server. These tables are not user tables (tables in which users store business data). The extenders use them 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 145.

When you issue the DBxEnableServer API or ENABLE SERVER command, you:

- *Should* specify a table space to hold “global” administrative support tables for the database server. These administrative support tables store information such as for which extenders the database server is enabled. If you enable a database server for the Image Extender, one of the global administrative support tables records the association between user table columns and QBIC catalogs.
- *Must* specify an MVS Workload Manager (WLM) environment name (you can specify two). UDFs for the extender run in these WLM environments.
- *Can* specify external security. This indicates how the UDFs interact with an external security product, such as IBM Resource Access Control Facility (RACF) and the file system, to control access to files. UDFs that use files include UDFs that store, retrieve, and update objects, such as DB2Image and Content, they do not include attribute retrieval UDFs such as Format.

You need SYSADM authority, a user ID of MMDBSYS, or a user ID with a secondary authorization ID of MMDBSYS to enable a database server.

Specifying table space

The table space specification has two parts. The first part is the table space name. The name must be the name of a table space that is defined in the MMDBSYS database. (The MMDBSYS database is created as part of the setup that is done after the DB2 extenders are installed.) For further information about DB2 extender

Enabling database servers

installation and setup procedures, see the *Program Directory*.) If you do not specify a table space name, DB2 creates a table space in the MMDBSYS database for each global administrative support table.

The second part of the table space specification identifies any combination of using-block, free block, gbpccache-block, and index options for type 2 non-partitioned indexes. You get defaults if you do not provide the second part of the table space specification. For details about these blocks and index options, see the description of the CREATE INDEX command in the *SQL Reference*.

Specifying WLM environments

You can specify up to two WLM environment names. (WLM environments for the DB2 extenders are established as part of the setup that is done after the DB2 extenders are installed.) If only one WLM environment name is specified, then all extender UDFs run in that WLM environment. If two WLM environments are specified, the second is used to execute UDFs that store, retrieve, or update objects (such as DB2Image, Content, and Replace). The first WLM environment is used for attribute retrieval UDFs (such as Width, Height, and Size).

See “Workload management considerations” on page 31 for additional considerations in specifying WLM environments.

Specifying external security

You can specify EXTERNAL SECURITY USER or EXTERNAL SECURITY DB2. If you specify EXTERNAL SECURITY USER, each UDF runs as if has the user ID (that is, the primary authorization ID) of the process that invoked it. Each UDF has permissions as defined for that user ID on the OS/390 server.

If you specify EXTERNAL SECURITY DB2, UDF access to files is performed using the primary authorization ID of the DB2 extenders. In this case, any files that UDFs can access using the extender authorization ID can also be accessed by the process that starts the UDF. EXTERNAL SECURITY DB2 is the default.

See “Security considerations” on page 32 for additional considerations in specifying external security.

Examples

In the following examples, a database server is enabled to hold image data.

Using the API: The code in Figure 14 on page 43 connects to an existing database server 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 member of the SAMPLES partitioned data set, and in ENABLE.C file in the SAMPLES Open Edition subdirectory.

```

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

#define MMDB_ERROR_MSG_TEXT_LEN 1000
#define SERVER_IS_DB2390 (strcmp(dbms,"DB2")==0 || strcmp(dbms,"DSN06010")==0)

int
main(int argc, char *argv[])
{
    SQLHENV henv = SQL_NULL_HENV;
    SQLHDBC hdbc = SQL_NULL_HDBC;
    SQLHSTMT hstmt = SQL_NULL_HSTMT;
    SQLCHAR uid[18+1];
    SQLCHAR pwd[30+1];
    SQLCHAR dbname[SQL_MAX_DSN_LENGTH+1];
    SQLCHAR buffer[500];
    SQL SMALLINT dbms_sz = 0;
    char dbms[20];

    SQLRETURN rc = SQL_SUCCESS;
    SQLINTEGER sqlcode = 0;
    char errorMsgText[MMDB_ERROR_MSG_TEXT_LEN+1];
    char *program = "enable;
    char tableSpace[8+1]="MMDBSYS"; /* define global meta tablespace */
    char wlm[8+1]="WLMENV1" /* define wlm environment */
    char security[8+1]="DB2"; /* define external security */

```

Figure 14. Sample code that enables a database server (Part 1 of 3)

Enabling database servers

```
/*--- Prompt for subsystem location name, wlm environment name, userid, --*/
/*--- and password -----*/
if (argc > 5) || (argc >= 2 && strcmp(argv[1], "?") == 0)
{
    printf("Syntax for enable - enabling a DB2 for OS/390 server: \n"
           "    enable location_name wlm_environment userid password\n");
    exit(0);
}

if (argc == 5) {
    strcpy((char *)dbname, argv[1]);
    strcpy((char *)wlm, argv[2]);
    strcpy((char *)uid, argv[3]);
    strcpy((char *)pwd, argv[4]);
}
else {
    printf("Enter DB2 location name:\n");
    gets((char *) dbname);
    printf("Enter userid:\n");
    gets((char *) uid);
    printf("Enter password:\n");
    gets((char *) pwd);
}

/*--- connect to DB2 for OS/390 server -----*/
rc = cliInitialize(&henv, &hdbc, dbname, uid, pwd);
cliCheckError(henv, hdbc, SQL_NULL_HSTMT, rc);
if (rc < 0) goto SERROR;

/*--- find out if application is connected to DB2/OS390? -----*/
rc = SQLGetInfo(hdbc, SQL_DBMS_NAME, (SQLPOINTER) &dbms,
               sizeof(dbms), &dbms_sz);
cliCheckError(henv, hdbc, SQL_NULL_HSTMT, rc);
if (rc < 0) goto SERROR;
```

Figure 14. Sample code that enables a database server (Part 2 of 3)

```

/***** enable server for image extender *****/
if (SERVER_IS_DB2390)
{
    printf("Enter WLM Environment Name:\n");
    gets((char *) wlm);          /* check later */
    printf("%s: Enabling server.....\n", program);
}
printf("%s: This may take a few minutes, please wait.....\n", program);

if (SERVER_IS_DB2390)
{
    printf("Enable server for db2image using %s wlm environment %s"
           " external security %s\n",
           program, tableSpace, wlm, security);
    rc = DBiEnableServer(tableSpace, wlm, security);
    step="DBiEnableServer"
}
if (rc < 0) {
    printf("%s: %s failed!\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    if (sqlcode)
        printf("sqlcode=%i, ", sqlcode);
    printf("errorMsgText=%s\n", errorMsgText);
} else if (rc > 0) {
    printf("%s: %s, warning detected.\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    printf("warning MsgText=%s\n", errorMsgText);
} else
    printf("%s: %s OK\n", program, step);
/***** end of enable server *****/

```

Figure 14. Sample code that enables a database server (Part 3 of 3)

Using the db2ext command line:

```

enable server for db2image using mmdbsysg wlm environment wlmenv1 external
security db2

```

Enabling tables

Use the DBxEnableTable API (where x is a for audio, i for image, or v for video) or the ENABLE TABLE command to enable a table for a DB2 extender.

When you issue the API or command, you specify table spaces to hold administrative support tables and LOB data for image, audio, and video objects. You also specify the name of the user table.

The table space specification has four parts:

- The name of the table space for the administrative support tables. You must specify this table space.
- For the index created on the administrative support tables, any combination of the using-block, free block, gbpcache-block, and index options for type 2 non-partitioned indexes. You get defaults if you do not specify this part of the table space specification.
- The name of the table space for LOB data. You must specify this table space.
- For the index created on the LOB table, any combination of the using-block, free block, gbpcache-block, and index options for type 2 non-partitioned indexes. You get defaults if you do not specify this part of the table space specification.

Enabling tables

For details about the blocks and options for indexes, see the description of the CREATE INDEX command in the *SQL Reference*.

It is recommended that you specify table spaces that are in the same database as the user table.

To enable a table, you need either:

- SYSADM authority
- DBADM authority with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- A user ID of MMDBSYS with a secondary authorization ID of MMDBSYS; the MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table. The SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

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

Using the API: In Figure 15 on page 47, before enabling the table, the code creates the table and table spaces. The example includes some error-checking code. The complete sample program is in the ENABLE member of the SAMPLES partitioned data set, and in ENABLE.C file in the SAMPLES Open Edition subdirectory.

```

char tableName[8+18+1] = "sobay_catalog
SQLCHAR szCreate_DB2390[]="CREATE TABLE %(s mmdbsys.DB2Image,
%(s mmdbsys.DB2Video, %(s mmdbsys.DB2Audio, artist varchar(25),
title varchar(25) stock_no char(11), tw char(10), price char(10))";

SQLCHR stmt02[]="CREATE TABLESPACE %(s SEGSIZE 8 BUFFERPOOL BP32K %(s";
SQLCHR stmt03[]="CREATE LOB TABLESPACE %(s LOG NO %(s";
SQLCHR stmt04[]="USING STOGROUP SYSDEFLT PRIQTY 50 SECQTY 10" %(s;
char tsp[8+1]="SAMPS";          /* define tablespace */
char tspli[8+1]="SAMPLI";       /* define lob tablespace for image */
char tsplblk[255]="";           /* define template for enabling */
/* table ('tsp,using blk, */
/* lob tsp, lob inx using blk) */

/*-----create table -----*/
printf("%s: Creating table .....\\n", program);
if (SERVER_IS_DB2390)
    sprintf((char*) buffer, (char*) szCreate_DB2390,
            tableName, imageColumn, videoColumn, audioColumn);

rc = SQLAllocStmt(hdbc, &hstmt);
cliCheckError(SQL_NULL_HENV, hdbc, SQL_NULL_HSTMT, rc);
rc = SQLExecDirect(hstmt, buffer, SQL_NTS);
cliCheckError(SQL_NULL_HENV, SQL_NULL_HDBC, hstmt, rc);

/*---- create tablespaces -----*/
if (SERVER_IS_DB2390)
{
    printf("%s:Creating tablespace %(s .....\\n", program, tsp);
    sprintf((char*) buffer, (char*) stmt02, tsp, (char*) stmt04);
    rc= SQLExecDirect(hstmt, buffer, SQL_NTS);
    cliCheckError(SQL_NULL_HENV, SQL_NULL_HDBC, hstmt, rc);

    /*---- create lob tablespace for image -----*/
    printf("%s:Creating LOB %(s for image .....\\n",
            program, tsp);
    sprintf((char*) buffer, (char*) stmt03, tspli, (char*) stmt04);
    rc= SQLExecDirect(hstmt, buffer, SQL_NTS);
    cliCheckError(SQL_NULL_HENV, SQL_NULL_HDBC, hstmt, rc);

    /*---- commit changes to database -----*/
    rc= SQLTransact(henv, hdbc, SQL_COMMIT);
    cliCheckError(henv, hdbc, SQL_NULL_HDBC, hstmt, rc);
}
/*---- end of create tablespaces -----*/

```

Figure 15. Sample code that enables a table (Part 1 of 2)

Enabling tables

```
/*---- enable table for image extender -----*/
printf("%s: Enabling table.....\n", program);
step="DBiEnableTable";
if (SERVER_IS_DB2390)
{
    sprintf((char*) tsplb, "%s, %s, %s, %s",
            tsp, (char*)stmt04, tspli, (char*) stmt04);
    rc = DBiEnableTable(tsplb, tableName);
}
if (rc < 0) {
    printf("%s: %s failed!\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    if (sqlcode)
        printf("sqlcode=%i, "sqlcode");
    printf("errorMsgText=%s\n", errorMsgText);
} else if (rc > 0) {
    printf("%s: %s, warning detected.\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    printf("warningMsgText=%s\n", errorMsgText);
} else
    printf("%s: %s OK\n", program, step)
/*---- end of enable table -----*/
```

Figure 15. Sample code that enables a table (Part 2 of 2)

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

```
enable table employee for db2image using tbspace1,,ltspace1
```

Enabling columns

Use the DBxEnableColumn API (where x is a for audio, i for image, or v for video) or the ENABLE COLUMN command to enable a column for a DB2 extender. When you issue the API or command, you specify the pertinent table and column.

When you enable a column, the extender adds information to the administrative support tables that belong to the user table.

To enable a column, you need either:

- SYSADM authority
- DBADM authority with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- A user ID of MMDBSYS with a secondary authorization ID of MMDBSYS; the MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table. The SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

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

Using the API: This example includes some error-checking code. The complete sample program is in the ENABLE member of the SAMPLES partitioned data set, and in ENABLE.C file in the SAMPLES Open Edition subdirectory.


```

char imageColumn[18+1] = "covers";

/*---- enable column for image extender ----*/
printf("%s: Enabling columns.....\n", program);
step="DBiEnableColumn";
rc = DBiEnableColumn(tableName, imageColumn);
if (rc < 0) {
    printf("%s: %s failed!\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    if (sqlcode)
        printf("sqlcode=%i, ", sqlcode);
    printf("errorMsgText=%s\n", errorMsgText)

} else if (rc > 0) {
    printf("%s: %s, warning detected.\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    printf("warningMsgText=%s\n", errorMsgText);
} else
    printf("%s: %s OK\n", program, step);
/*---- enable column for image extender ----*/

```

Figure 16. Sample code that enables a column

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

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

Disabling data objects

If you remove extender data from a database server, 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 15. Administration commands for the client” on page 351. For more information about the extender APIs, see “Chapter 14. Application programming interfaces” on page 215.

Before dropping a table or database server that contains extender data, disable it.

Disabling

Chapter 7. Tracking data objects and media files

As you create and debug applications that use the DB2 extenders, it is 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 is also useful to understand the correlation between user tables and external media files, for example, which tables refer to a specific file or which files are referenced by a specific table. It is also useful to discover if your tables refer to files that no longer exist on the system.

You need appropriate privileges: You need to have access to a table in order to track data in the table. If you want to perform comprehensive tracking operations, such as find which entries in all user tables in the database server refer to a file, you need SYSADM authority, DBADM authority, or SELECT privilege on enabled columns in all searched user tables and associated administrative support tables. If you do not have access to all the tables, the extenders will return tracking information only for those tables that you can access. They will also return a code indicating that you do not have access authority to some of the required tables.

Checking the status of data objects

You can use the DBXIsServerEnabled API or the GET EXTENDER STATUS command to check whether a database server is enabled for an extender. The following example determines if the current database server is enabled for the Image Extender. The database server is already connected. The complete sample program is in the API member of the SAMPLES partitioned data set, and in API.C file in the SAMPLES Open Edition subdirectory.

Using the API: The sample code in Figure 17 on page 52 includes some error-checking code.

Checking for enablement

```
/*---- Query the database server using DBIsServerEnabled API. -----*/
step="DBIsServerEnabled API";
rc = DBIsServerEnabled(&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 server 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 server 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 17. Sample code that checks if a database server 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 server. 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. Use the DBxAdminIsFileReferenced API to check which entries in all or a subset of user tables in the current database server refer to an external media file. Use the DBxIsFileReferenced API to check which entries in a specific user table refer to an external media file.

Using the API: The sample code in Figure 18 on page 53 returns the number of times a file is referenced and where it is referenced. It includes some error-checking code. The complete sample program is in the API member of the SAMPLES partitioned data set, and in API.C file in the SAMPLES Open Edition subdirectory.

```

/*---- Query the database server using DBAdminIsFileReferenced API. -----*/
step="DBAdminIsFileReferenced API";
rc = DBAdminIsFileReferenced((char*) uid, filename, &count, &filelist);
if (rc < 0) {
    printf("%s: %s FAILED!\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else if (rc > 0) {
    printf("%s: %s, warning detected.\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else {
    if (count == 0)
        printf("%s: \"%s\" file is not referenced\n",
            program, filename);
    else {
        printf("%s: \"%s\" file is referenced %d times\n",
            program, filename);
        for (i=0; i < count; i++)
        {
            /* filename is NULL for any IsFileReferenced APIs */

            printf ("filename = %s\n", filelist[i].filename);
            printf ("\tqualifier = %s\n", filelist[i].tqualifier);
            printf ("\ttable = %s\n", filelist[i].tname);
            printf ("\thandle = %s\n", filelist[i].handle);
            printf ("\tcolumn = %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 18. Sample code that checks if a file is referenced by user tables

Finding files referenced by table entries

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

Using the API: The sample code in Figure 19 on page 54 returns the number of files it finds and a list of the files. The complete sample program is in the API member of the SAMPLES partitioned data set, and in the API.C file in the SAMPLES Open Edition subdirectory.

Listing referenced files

```
/*---- Query the database using DBiAdminGetReferencedFiles API. -----*/
step="DBiAdminGetReferencedFiles API"
rc = DBiAdminGetReferencedFiles((char*) uid, &count, &filelist);
if (rc < 0) {
    printf("%s: %s FAILED!\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else if (rc > 0) {
    printf("%s: %s, warning detected.\n", program, step);
    printMsg(rc);
    DBiGetError(&sqlcode, errorMsgText);
    printf("sqlcode=%i, errorMsgText=%s\n", sqlcode, errorMsgText);
} else {
    if (count == 0)
        printf("%s: no referenced files\n", program);
    else {
        printf("%s: %d referenced files\n", program, count);
        for (i=0; i < count; i++)
        {
            printf ("filename = %s\n", filelist[i].filename);
            printf ("\tqualifier = %s\n", filelist[i].tqualifier);
            printf ("\tttable = %s\n", filelist[i].tname);
            printf ("\thandle = %s\n", filelist[i].handle);
            printf ("\tcolumn = %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 19. 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 that 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.

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 server. Use the DBxGetInaccessibleFiles API or the GET INACCESSIBLE FILES command to list the inaccessible media files that are referenced by a specific table.

Chapter 8. Granting and revoking privileges on administrative support tables

Users issue UDFs to select, insert, update, or delete image, audio, and video objects from a user table. To perform the requested operations, the UDFs must be able to access, and if necessary insert, update, and delete in, the administrative support tables that hold attribute information for the objects. For the owner of a user table, the extenders automatically give the UDFs the access they need to handle the requested operation. However users other than the table owner who need to select an object from the user table must be granted select privilege on the administrative support tables.

In addition, users who perform QBIC operations on image objects in a user table must have appropriate privileges on the administrative support tables that comprise the QBIC catalog for those objects. For example a user who issues a QBIC query against a column of images must have SELECT privilege on the QBIC catalog tables for the image column. A user who makes changes to the QBIC catalog should have SELECT, INSERT, UPDATE, and DELETE privilege on the associated QBIC catalog tables.

The owner of a user table or a DBA (with GRANT privilege) for the database can use the DB2 extender command GRANT to grant privileges on the administrative support tables. When you issue the GRANT command, you specify:

- The required privilege, for example, SELECT or UPDATE.
- The name of the extender: DB2IMAGE, DB2AUDIO, or DB2VIDEO. You can also specify ALL for all three extenders.
- The name of the user table.
- The ID of the user. You can precede the user ID with the optional keyword USER. You can also specify PUBLIC for all users.

If you specify SELECT, you grant SELECT privilege to the specified users on the administrative support tables for the named extenders that are associated with the user table. If you specify DB2IMAGE, you also grant SELECT privilege on the administrative support tables for the QBIC catalogs associated with the user table. For example, the following command grants SELECT privilege on the administrative support tables for the Image Extender that are associated with the employee table. The privilege is granted to user ID ajones. The command also grants to user ID ajones SELECT privilege on the QBIC catalogs associated with the employee table:

```
grant select for db2image on employee to ajones
```

The following command grants SELECT privilege on the administrative support tables for the Image, Audio, and Video Extenders that are associated with the employee table. The privilege is granted to all users. The command also grants to all users SELECT privilege on the QBIC catalogs associated with the employee table:

```
grant select for all on employee to public
```

For insert, update, or delete operations, the extenders check to determine if the user has the needed INSERT, UPDATE, or DELETE privilege on the user table. If

the user has the required privilege, the extenders allow the UDFs to access the administrative support tables, as required.

To grant INSERT, UPDATE, and DELETE privileges on the QBIC Catalog tables, specify UPDATE and DB2Image in the GRANT command. For example, the following command grants INSERT, UPDATE, and DELETE privileges to user ID ajones on the QBIC Catalog tables associated with the employee table:

```
grant update for db2image on employee to user ajones
```

When it is no longer appropriate for a user to access an object in a user table, the owner of the user table or a DBA (with GRANT privilege) for the database can revoke the user's SELECT privilege on the administrative support tables. This also includes administrative support tables that comprise QBIC catalogs. Use the DB2 extender command REVOKE to revoke privileges on the administrative support tables and QBIC catalog tables. The format of the REVOKE command is similar to the GRANT command. For example, the following command revokes SELECT privilege on the administrative support tables for the Image Extender associated with the employee table. The privilege is revoked for user ID ajones. The command also revokes SELECT privilege on the QBIC Catalog tables associated with the employee table:

```
revoke select for db2image on employee from ajones
```

You can also revoke INSERT, UPDATE, and DELETE privileges on the administrative support tables that comprise QBIC catalogs. Use the UPDATE parameter on the REVOKE command. For example, the following command revokes INSERT, UPDATE, and DELETE privileges on the QBIC Catalog tables associated with the employee table. The privileges are revoked for user ID ajones.

```
revoke update for db2image on employee from ajones
```

Grant privileges on a QBIC catalog after all features are added: Privileges granted on administrative support tables that comprise a QBIC catalog include privileges on QBIC features tables, but only for features that have been already been added to the catalog. If you add a feature to the catalog after you grant privileges on the catalog, you will have to grant privileges on the catalog again. So you should grant privileges on a QBIC catalog only after the catalog is created and after all the features have been added.

Part 3. Programming for image, audio, and video data

Chapter 9. Programming overview	59
Using extender UDFs and APIs.	59
Tasks you can perform with extender UDFs and APIs.	60
Sample table for extender examples	60
Before you begin programming for DB2 extenders	61
Including extender definitions	63
Specifying UDF and UDT names	63
Transmitting large objects.	64
If the object is transmitted between a table and a server file	64
If the object is transmitted to or from a client buffer	64
Using LOB locators.	65
If the object is transmitted to or from a client file	65
Specifying file names when you transmit objects	66
Handling return codes.	67
Preparing a DB2 extender application	67
Preparing a DB2 application.	67
Additional steps for DB2 extender applications	68
Binding files for workstation clients	68
Including MMBSYS_CLIENT packages.	69
Using the DSNALI call to allocate resources	69
Configuring the ODBC Initialization file.	69
Unicode support.	70
 Chapter 10. Storing, retrieving, and updating objects	71
Image, audio, and video formats	71
Image conversion options	72
Storing an image, audio, or video object.	73
DB2Image, DB2Audio, and DB2Video UDF formats.	74
DB2ImageA, DB2AudioA, and DB2VideoA UDF formats.	76
Storing an object that resides on the client	77
Storing an object that resides on the server	78
Specifying database or file storage.	78
Identifying the format for storage	79
Identifying the format for storage without conversion.	79
Identifying the formats and conversion options for storage with format conversion.	80
Storing an object with user-supplied attributes.	81
Storing a thumbnail (image and video only)	82
Storing a comment	83
Retrieving an image, audio, or video object.	84
Content UDF formats for retrieval.	84
Retrieving an object to the client	85
Retrieving an object to a client without format conversion.	85
Retrieving an image to a client with conversion.	86
Retrieving an object to a server file	87
Retrieving and using attributes	88
Retrieving comments	90
Updating an image, audio, or video object	90
Content UDF formats for updating	91
ContentA UDF formats for updating	92
Replace UDF formats for updating	93
ReplaceA UDF formats for updating	95
Updating an object from the client.	96
Updating an object from the server	97
Specifying database or file storage for updates	97
Identifying the format for update	98
Identifying the format for update without conversion.	98
Identifying the formats and conversion options for update with format conversion	99
Updating an object with user-supplied attributes	99
Updating a thumbnail (image and video only)	100
Updating a comment	101
 Chapter 11. Displaying or playing an image, audio, or video object	103
Using the display or play APIs	103
Identifying a display or play program	103
Specifying BLOB or file content	104
Specifying a wait indicator	105
Displaying a thumbnail-size image or video frame	105
Displaying a full-size image or video frame	106
Playing an audio or video	106
 Chapter 12. Querying images by content	107
How to query by image content	107
Managing QBIC catalogs	108
Creating a QBIC catalog.	108
Opening a QBIC catalog.	110
Adding a feature to a QBIC catalog	111
Removing a feature from a QBIC catalog	112
Retrieving information about a QBIC catalog	112
Manually cataloging a column of images	114
Recataloging images	114
Closing a QBIC catalog	115
Deleting a QBIC catalog.	115
QBIC catalog sample program.	115
Building queries	122
Specifying a query string	123
Feature value	123
Feature weight	124
Examples.	124
Using a query object	125
Editing and running job DMBSETUP	125
Creating a query object	126
Adding a feature to a query object	126
Specifying the data source for a feature in a query object.	126
Setting the weight of a feature in a query object	129
Saving and reusing a query string	129

Retrieving information about a query object	130
Removing a feature from a query object . . .	131
Deleting a query object	131
Issuing queries by image content.	131
Querying images	132
Retrieving an image score	133
Retrieving the score of a single image . . .	133
Retrieving the score of multiple images. . .	134
QBIC query sample program	134

Chapter 9. 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 server. 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 run 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 server*/
        '/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 run in the client.

The following C statements include an API that is named DBiBrowse. The API retrieves the data for an image handle and starts 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*/
```

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 73
Retrieve an image, audio, or video object	Page 84
Retrieve and use image, audio, and video attributes	Page 88
Retrieve comments associated with an image, audio, or video object	Page 90
Update an image, audio, or video object	Page 90
Display an image object	Page 103
Display a thumbnail-size image or video frame	Page 105
Play an audio or video object	Page 106
Query images by content	Page 107

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 that is named `EMPLOYEE`, and that the table 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 20 on page 61 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  
);
```

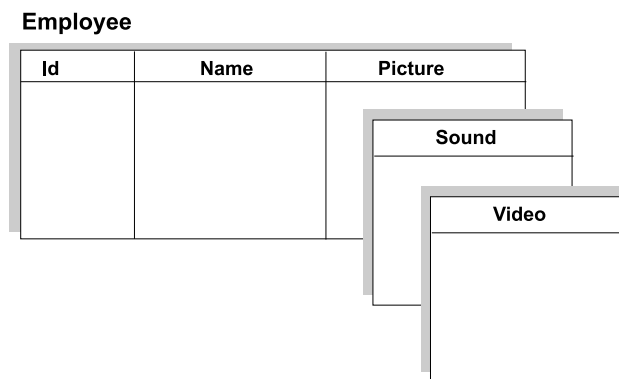


Figure 20. A table used in DB2 extender programming examples

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 Application Programming and SQL 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 that are defined by the extenders. For example, Figure 21 on page 62 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 that is 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.

Before you begin

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[17];            /* database server name */
    char hvName[40];              /* employee name */
    char hvFormat[9];             /* image format */
    long hvSize;                  /* image size */
    long hvWidth;                 /* image width */
    long hvHeight;                /* image height */
    char hvComment[16385]
    short indComment;
EXEC SQL END DECLARE SECTION;

* Set current path
*/
EXEC SQL SET CURRENT PATH = mmdbsys, CURRENT PATH;
```

Figure 21. An application that uses a DB2 extender (Part 1 of 2)

```

/*
 * 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:indComment;

    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);
    printf("comment = %s\n", hvComment);
}

/*
 * 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 21. 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 that are used by the extender. The include files are provided as HFS files, and in a header file partitioned data set. The names of the include files are:

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 that is created by the Image Extender is `mmdbsys.DB2Image`. You can omit the `mmdbsys` schema-name if you previously set the current path to `mmdbsys`, for example:

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

Before you begin

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. The method you use also depends on whether the file is in your client machine or in the database server machine. You can transmit an object to a client file only if the file is in a workstation client.

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 that are 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 that is 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 VARCHAR(254)),
'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 the LOB 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 that is 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';
```

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 workstation. (If the client file is in an OS/390 client, you can copy the content of the file to a buffer and then transmit it. You can also transmit the content to a buffer and then copy to the client file.) 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:

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).

Before you begin

Options for output:

SQL_FILE_CREATE. This option creates a new file if it does not 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 that is added to the file (in bytes), not the total length of the file.

For example, the following statements declare a file reference variable that is 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 VARCHAR(254)),
        '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.

DB2 extenders give you access to files in a file system that is compatible with OS/390 UNIX services, for example, a hierarchical file system (HFS). 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 a file system such as HFS, a relative file name is any file name that does not begin with a slash.

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 typically 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 409 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 an OS/390 UNIX

file system. For example, a Windows NT file name such as `c:\dir1\abc.bmp` is converted to `/dir1/abc.bmp` when passed to the server.

Handling return codes

All embedded SQL statements or DB2 CLI calls in your program, including those that request DB2 extender UDFs, generate codes that indicate whether the embedded SQL statement or DB2 CLI call ran 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 that are returned by embedded SQL statements, CLI calls, and APIs.

For information on handling these return codes, see “Chapter 16. Diagnostic information” on page 381.

In situations where an extender API cannot successfully complete its unit of work, a rollback operation is performed. The API also returns an error code. The rollback operation is done so that the database can be returned to its previous consistency point. Refer to “Chapter 14. Application programming interfaces” on page 215 for details.

Preparing a DB2 extender application

Like all DB2 programs, application programs that use DB2 extender UDFs or APIs must be prepared before they can be used. In general, you prepare a DB2 extender application in the same way as you prepare any DB2 application. However there are some additional steps that you need to perform.

You can prepare an application that uses DB2 extender UDFs or APIs on a workstation client or on an OS/390 client. The steps you follow depend on whether your program runs in a DB2 ODBC environment. If your program does run in a DB2 ODBC environment, it can include CLI calls. If your program does not run in a DB2 ODBC environment, it can include embedded SQL statements. In either case, your program can include calls to DB2 extender APIs.

Preparing a DB2 application

To prepare a DB2 application program, you perform the following steps:

- Precompile the source files for the program. Precompiling creates a modified version of your program. The SQL statements in the program are replaced with C or C++ language inserts. This makes the program compatible with the compiler. Precompiling also creates information about the SQL statements in the program. If you precompile the program on a workstation client, the information about the SQL statements is placed in a bind file. If you precompile the program on an OS/390 client, the information about the SQL statements is placed in a Database Request Module (DBRM).
- Bind the bind file or bind the DBRM. Connect to the database server before you do the bind. The bind produces a plan. The database server uses information in the plan to satisfy SQL requests made by the application program. You can also bind a DBRM to a package. You can then bind the package to a plan, including other packages and DBRMs.
- Compile the modified source files. This creates an object module. When you precompile a DB2 extender application, specify the `DLL`, `LONGNAME`, and `RENT` options.
- Prelink and link edit the object module. Specify any needed DLLs or shared libraries. This can include extender client libraries or ODBC libraries. How the

Before you begin

DLLs or shared libraries are specified depends on the platform. On most UNIX platforms, the DLLs or shared libraries are specified by name. In Windows environments, the DLLs or shared libraries are specified in a lib file. In OS/390, the DLLs or shared libraries are specified in export files.

When you prelink and link edit on an OS/390 client, specify the options that will generate the following executable module options: RMODE(ANY), AMODE(31), and RENT.

For further information about preparing on a workstation client a DB2 application program that does not run in a DB2 ODBC environment, see *DB2 Universal Database Application Development Guide, Version 6*. For further information about preparing on an OS/390 client a DB2 application program that does not run in a DB2 ODBC environment, see *DB2 Universal Database for OS/390 Version 6 Application Programming and SQL Guide*.

For further information about preparing on a workstation client a DB2 application program that runs in a DB2 ODBC environment, see *DB2 Universal Database CLI Guide and Reference, Version 6*. For further information about preparing on an OS/390 client a DB2 application program that runs in a DB2 ODBC environment, see *DB2 Universal Database for OS/390 Version 6 ODBC Guide and Reference*.

Additional steps for DB2 extender applications

In addition to the general steps outlined in “Preparing a DB2 application” on page 67, there are other steps you need to take to prepare and run a DB2 extender application.

Binding files for workstation clients

The DB2 extenders client code for workstation clients have associated bind files that you need to bind to the DB2 server. The bind files for each client platform are grouped together in a listfile, that is, a file with the extension .lst. To bind the files, connect to the database server and specify the bind command as follows:

```
bind path/@listfile grant public isolation cs
```

where *path* is the full path name of the directory in which the listfile is located. The paths are:

Path	Client
/usr/lpp/db2ext/lib	AIX
install_path/bin (the default install path is c:\dmb)	Windows NT, 98, 95
/opt/IBMdbs2ex/V6.1/lib	Solaris Operating Environment

and *listfile* is the name of the listfile. The listfiles are:

Listfile	Client
dmbmvsb1.lst	AIX
dmbmvsb3.lst	Windows NT, 98, 95
dmbmvsb8.lst	Solaris Operating Environment

For example, the following command binds the listfile for a Windows NT client:

```
bind c:\dmb\bin@dmbmvsb3.lst grant public isolation cs
```

Including MMDBSYS_CLIENT packages

If your OS/390 application includes calls to DB2 extender APIs, you need to include in your plan the packages comprised by MMDBSYS_CLIENT.*. If your application does not include any embedded SQL statements or does not include CLI calls, use the plan DMBAPI when you run your application. DMBAPI includes the packages that comprise MMDBSYS_CLIENT.*. If your application does include CLI calls, use the plan DMBACLI when you run your application.

Using the DSNALI call to allocate resources

If your application includes embedded SQL statements and you do not use ODBC to connect, you need to attach your application to the DB2 subsystem. There are various ways to attach to the DB2 subsystem. One way is to use the Call Attachment Facility (CAF). CAF is compatible with OpenEdition and with TSO environments. Use the CAF Open function, DSNALI, to allocate resources, and to request a DB2 connection. In the DSNALI call, you specify the appropriate plan name as follows:

```
dsnali ("OPEN", "subsystem_name", "planname", rc, reascode)
```

where *subsystem_name* is the name of the DB2 subsystem, *planname* is the plan name, *rc* is the return code, and *reascode* is the reason code.

For example, the following DSNALI call allocates resources for plan name MYPLAN in subsystem V61A:

```
dsnali("OPEN", "V61A", "MYPLAN", rc, reascode)
```

Configuring the ODBC Initialization file

If your application runs in a DB2 ODBC environment on OS/390, you need to do either of the following:

- Configure the ODBC INI (initialization) file, and override the plan name in the subsystem stanza for the initialization file.
- Specify the plan name in the SQLDriverConnect call.

To configure the ODBC INI file in TSO, specify the DSNAOINI DD card in the JCL or CLIST that starts your application. For example, the following CLIST command specifies the DSNAOINI DD card for an ODBC INI file named cli.ini:

```
ALLOCATE DD(DSNAOINI), DA(cli.ini) SHR
```

To configure the ODBC INI file for the UNIX shell, you need to specify an environment variable, DSNAOINI, for example:

```
export DSNAOINI = ./cli.ini
```

You need to ensure that you override the plan name in the subsystem stanza within the ODBC INI file, and not in the common or datasource stanza. For example, the following specifies the plan name DMBACLI for the subsystem stanza V61A:

```
[common]
MVSDEFAULTSSID = V61A
[V61A]
PLANNAME = DMBACLI
```

The following specifies the plan name DMBACLI in the SQLDriverConnect call:

Before you begin

```
SQLAllocEnv  
SQLAllocConnect  
SQLDriverConnect (hdbc  
                  NULL  
                  "planname = dmbac1i", ...)
```

Unicode support

Observe the following points regarding Unicode support for the Image, Audio, and Video Extenders:

- The only parameters that can be a Unicode string are the comment fields in the following UDFs:
 - `mmdbsys.db2image()` import an image
 - `mmdbsys.db2audio()` import an audio
 - `mmdbsys.db2video()` import a video
 - `mmdbsys.replace()` replace an image, an audio, or a video
 - `mmdbsys.comment()` comment update
- If you are planning to access an Unicode database, you must use a DB2 extenders instance set up to support Unicode. An Unicode instance will only handle Unicode database.

In order for an extender instance to support Unicode, you set the environment variable `DB2CODEPAGE` to 1208 before invoking `DMBSTART`.

Chapter 10. 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	
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	

Formats

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

Format	Description	Read	Write
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 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.

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 changes the interpretation of an image, based on the value specified. It does not change the image itself. This option applies to black and white or grayscale images only, and does not apply to images in GIF format.	0 = Ones are black 1 = Ones are white
-n	Photometric (image inversion). This option changes an image by inverting black to white, and white to black. The option applies to black and white or grayscale images only.	None

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 (continued)

Parameter	Description	Value
-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 (the client file must be in a workstation client) 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:

- The name of the currently connected database server; this is contained in the CURRENT SERVER special register.
- The source of the object content; this is either in a client buffer, client file (workstation client only), 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 do not want to store a comment).

The Image, Audio, and Video Extenders allow you to store an object even if they do not recognize the object's format. Use the DB2ImageA, DB2AudioA, or DB2VideoA UDF in an SQL INSERT statement to store an image, audio, or video object with an unrecognized format in a database. You need to specify the attributes of the object, its format, and for video objects only, its compression format. When you store an image or video with user-supplied attributes, you can also store a thumbnail. A thumbnail is a miniature image representing the image or video.

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

Storing

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 workstation client file:

```
DB2xxxxx(  
    CURRENT SERVER,      /* database server 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 server 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 */  
);
```

The DB2Image UDF includes the following additional formats:

Format 3: Store an image from a client buffer or workstation client file with format conversion:

```
DB2Image(  
    CURRENT SERVER,      /* database server 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 4: Store an image from a server file with format conversion:

```
DB2Image(  
    CURRENT SERVER,      /* database server 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 5: Store an image from a client buffer or workstation client file with format conversion and additional changes:

```

DB2Image(
  CURRENT SERVER,      /* database server 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 6: Store an image from a server file with format conversion and additional changes:

```

DB2Image(
  CURRENT SERVER,      /* database server 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 that is 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 server*/
    '/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 4 above.)

```

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

```

Storing

```
'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. 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.

DB2ImageA, DB2AudioA, and DB2VideoA UDF formats

The DB2ImageA, DB2AudioA, and DB2VideoA 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 with user-supplied attributes from a client buffer or workstation client file:

```
DB2xxxxx(
    CURRENT SERVER,    /* database server 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 */
    tracknames,        /* MIDI track names (audio only) */
    instruments,       /* MIDI instrument names (audio only) */
    format,            /* source format */
    compress_type,     /* compression format (video only */
    thumbnail          /* thumbnail (image and video only) */
);
```

Format 2: Store an object with user-supplied attributes from a server file:

```
DB2xxxxx(
    CURRENT SERVER,    /* database server 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 */
    tracknames,        /* MIDI track names (audio only) */
    instruments,       /* MIDI instrument names (audio only) */
    format,            /* source format */
    compress_type,     /* compression format (video only */
    thumbnail          /* thumbnail (image and video only) */
);
```

For example, the following statements in a C application program store a row that includes an image 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.

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
char hvImgattrs[100];
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;
```

```

pimgattr = (DB2IMAGEATTRS *) hvImgattrs;
pimgattr->width=640;
pimgattr->height=480;

DBiPrepareAttrs(pimgattr);

DBEXEC SQL INSERT INTO EMPLOYEE VALUES(
    '128557',                                /* id */
    'Anita Jones',                          /* name */
    DB2IMAGEA(                              /* Image Extender UDF */
        CURRENT SERVER,                    /* database server */
        '/employee/images/ajones.bmp',    /* source file */
        :hvStorageType,                   /* stortype */
        'Anita''s picture', /* user comment */
        :hvImgattrs,                      /* user-specified attributes */
        'FormatI'                         /* source format */
    )
);

```

When you store an image, audio, or video 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 workstation client file to the server. (If the client file is in an OS/390 client, you can copy the content of the file to a buffer and then transmit it 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. The audio clip is stored 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',
        '',
        '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

Storing

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')
      );
```

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 409.

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 (workstation client only), 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 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 `VARCHAR(254)`.

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. The database table points to the server file:

```
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 71. 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. The target image 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

Storing

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 71. 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 71 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 72. 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. You must also specify the format of the object, and for video objects only, the compression format. Assign the attribute values to a VARCHAR(4096) FOR BIT DATA variable in the UDF. For MIDI audio objects only, you must also specify the tracknames and instruments for the MIDI audio. If the audio object is not MIDI, specify empty strings for the tracknames and instruments.

The UDF code on the server always expects data in “big endian format”. Big endian format is a format used by most UNIX and OS/390 platforms. If you are storing an object in “little endian format”, you need to prepare the user-supplied attribute data so that UDF code on the server can correctly process it. Little endian format is a format typically used in an Intel® and other microprocessor platform. (Even if you are not storing the object in little endian format, it is a good idea to prepare the user-supplied attribute data.) Use the DBiPrepareAttrs API to prepare attributes for image objects. Use the DBaPrepareAttrs API to prepare attributes for audio objects. Use 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;
char hvImgattrs[100];
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs;
pimgattr->width=640;
pimgattr->height=480;

DBiPrepareAttrs(pimgattr);

DBEXEC SQL INSERT INTO EMPLOYEE VALUES(
    '128557',
    'Anita Jones',
    DB2IMAGEA(
        CURRENT SERVER,
        '/employee/images/ajones.bmp',
        :hvStorageType,
        'Anita''s picture',
        :hvImgattrs,                                /* user-specified attributes */
        'FormatI',
        '')
    );

```

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

Storing

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;
char hvAudattr[100];
EXEC SQL END DECLARE SECTION;

MMDBAudioAttrs      *paudiattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

paudioattr=(MMDBAudioAttrs *) hvAudattr;
paudioattr->ulSamplingRate=44100;
paudioattr->usNumChannels=2;

DBaPrepareAttrs(paudioAttr);

EXEC SQL INSERT INTO EMPLOYEE VALUES(
    '128557',
    'Anita Jones',
    DB2AUDIOA(
        CURRENT SERVER,
        '/employee/sounds/ajones.aud',
        :hvStorageType,
        'Anita''s voice',
        :hvAudattr,
        '',
        '',
        'FormatA')
    );
```

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 and a compression format of MPEG1. 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;
    char hvVidattrs[4000];
    char hvThumbnail[16384];
EXEC SQL END DECLARE SECTION;

MMDBVideoAttrs      *pvideoAttr;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

pvideoAttr=(MMDBVideoAttrs *)hvVidattrs;

/* Generate thumbnail and assign to thumbnail variable */

EXEC SQL INSERT INTO EMPLOYEE VALUES(
    '128557',
    'Anita Jones',
    DB2VIDEOA(
        CURRENT SERVER,
        '/employee/videos/ajones.vid',
        :hvStorageType,
        'Anita''s video',
        :hvVidattrs,
        'FormatV',
        'MPEG1',
        :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 VARCHAR(16384). If you do not want a comment stored when you store an object, specify an empty string in place of the comment.

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 */
    );

```

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',

```

Storing

```
'Anita Jones',
DB2IMAGE(
  CURRENT SERVER,
  '/employee/images/ajones.bmp',
  'GIF',
  :hvStorageType,
  '') /* 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 (workstation client only), or server file.

Content UDF formats for retrieval

The Content UDF is overloaded, meaning, that 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 workstation client file:

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

Format 2: Retrieve a segment of an object to a client buffer or workstation 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 workstation 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 workstation 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, or retrieve the LOB. Use a file reference variable to retrieve an image, audio, or video object to a workstation client file. (For an OS/390 client, you can use a LOB locator or retrieve the LOB to a buffer and then write the LOB to a client file.)

Retrieving

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';
```

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 workstation client file with format conversion. (For an OS/390 client, you can use a LOB locator or retrieve the LOB to a buffer and then write the LOB to a client file.)

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 71 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 72 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 it is stored in a database table as a BLOB. The retrieved image is converted to GIF and it is 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 does not 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;
      char hvVid_fname[255];
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 71 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;
      char hvImg_fname[255];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT CONTENT(
      PICTURE,                                /* image handle */
      '/employee/images/ajones.gif',         /* target file */
      1,
      'GIF');
```

Retrieving

```
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. Alternatively, you must 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 that are 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 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. These attributes 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

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

Attribute	UDF	Image	Audio	Video
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	
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

Retrieving comments

Use the Comment UDF to retrieve comments that are 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 that is stored with an audio clip in the employee table.

```
EXEC SQL BEGIN DECLARE SECTION;
      char hvComment[16385];
      short indComment;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT COMMENT(SOUND)
      INTO :hvComment:indComment
      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;
      char hvImg_fname[255];
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 that is associated with the object. In either case, the extender updates the attributes that are 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 (workstation client only), or server file.

Table 5 on page 71 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 user specified the object's attributes when the object was stored. 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, that 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 workstation 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*/
);
```

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

Format 3: Update an image from a client buffer or workstation 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 4: 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 5: Update an image from a client buffer or workstation 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 6: Update an object from a server file with format conversion and additional changes:

Updating

```
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 that is 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 4 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';
```

ContentA UDF formats for updating

The ContentA UDFs are overloaded, that is, they have different formats depending on how the UDFs are used. The formats are as follows:

Format 1: Update an object with user-supplied attributes from a client buffer or workstation client file:

```
ContentA(  
    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 */  
    tracknames,           /* MIDI track names (audio only) */  
    instruments,          /* MIDI instruments (audio only) */  
    format,               /* source format */  
    compress_type,        /* compression format (video only) */  
    thumbnail             /* thumbnail (image and video only) */  
);
```

Format 2: Update an object with user-supplied attributes from a server file:

```
ContentA(  
    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 */
    tracknames,           /* MIDI track names (audio only) */
    instruments,          /* MIDI instruments (audio only) */
    format                /* source format */
    compress_type         /* compression format (video only) */
    thumbnail             /* thumbnail (image and video only) */
);

```

For example, the following statements in a C application program update an image in 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.

```

EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
char hvImgattrs[100];
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs;
pimgattr->width=640;
pimgattr->height=480;

DBiPrepareAttrs(pimgattr);

EXEC SQL UPDATE EMPLOYEE
SET VIDEO=CONTENTA(
    PICTURE,                /* image handle */
    '/employee/newimg/ajones.bmp', /* source file */
    :hvStorageType,        /* stotype */
    :ImgAttrs,             /* user-supplied attributes */
    'FormatI',             /* source format */
    '')                    /* no thumbnail */
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:

Format 1: Update an object from a client buffer or workstation 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 */
    stotype,               /* MMDB_STORAGE_TYPE_EXTERNAL=store */
                        /* in file server */
    comment                /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB */
                        /* user comment */
);

```

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

Updating

Format 3: Update an image from a client buffer or workstation 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 4: 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 */
    comment                /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB */
                          /* user comment */
);
```

Format 5: Update an image from a client buffer or workstation 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 6: 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 */
    comment                /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB */
                          /* 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 that is 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 4 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 */
```

ReplaceA UDF formats for updating

The ReplaceA UDFs are overloaded, that is, they have different formats depending on how the UDFs are used. The formats are as follows:

Format 1: Update an object with user-supplied attributes from a client buffer or workstation client file and update its comment:

```
ReplaceA(
    handle,                                /* object handle */
    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 */
    tracknames,                           /* MIDI track names (audio only) */
    instruments,                          /* MIDI instruments (audio only) */
    format                                /* source format */
    compress_type                          /* compression format (video only) */
    thumbnail                             /* thumbnail (image and video only) */
);
```

Format 2: Update an object with user-supplied attributes from a server file and update its comment:

```
ReplaceA(
    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*/
    comment                               /* user comment */
    attrs,                                /* user-supplied attributes */
    tracknames,                           /* MIDI track names (audio only) */
    instruments,                          /* MIDI instruments (audio only) */
    format                                /* source format */
    compress_type                          /* compression format (video only) */
    thumbnail                             /* thumbnail (image and video only) */
);
```

Updating

For example, the following statements in a C application program update a video clip stored in a server file, its thumbnail, and its associated comment.

```
EXEC SQL BEGIN DECLARE SECTION;
    long hvStorageType;
    char hvVidattrs[3500]
    char hvThumbnail[16384]
EXEC SQL END DECLARE SECTION;

MMDBVideoAttrs      *pvideoAttr;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

pvideoAttr=(MMDBVideoAttrs *)hvVidattrs;

EXEC SQL UPDATE EMPLOYEE
    SET VIDEO=REPLACEA(
        VIDEO,                                /* video handle */
        '/employee/newvid/ajones.mpg',      /* source file */
        :hvStorageType,                      /* stortype */
        'Anita's new video') /* user comment */
        :hvVidattrs,                          /* user-supplied attributes */
        'FormatV',                           /* source format */
        'MPEG1',                             /* compression format */
        :hvThumbnail)                        /* thumbnail */
    WHERE NAME='Anita Jones';
```

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 workstation client file. (If the client file is in an OS/390 client, you can copy the content of the file to a buffer and then transmit it to the server, or you can create a LOB locator and copy the file to the LOB locator.)

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

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. The database points to the 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 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 409.

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 (workstation client only), 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.

Updating

```
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 that is 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. The target image 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 71. 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 71 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 72. 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',
    'GIF',
    '')
  WHERE NAME='Anita Jones';
```

/*source format*/
/*target format*/

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',
    'GIF',
    '-r 1',
    '')
  WHERE NAME='Anita Jones';
```

/*source format*/
/*target format*/
/* conversion options */

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. You must also specify the format of the object, and for video objects only, the compression format. Assign the attribute values to a VARCHAR(4096) FOR BIT DATA variable in the UDF. For MIDI audio objects only, you must also specify the tracknames and instruments for the MIDI audio. If the audio object is not MIDI, specify empty strings for the tracknames and instruments.

The UDF code on the server always expects data in “big endian format”. Big endian format is a format used by most UNIX and OS/390 platforms. If you are storing an object in “little endian format”, you need to prepare the user-supplied attribute data so that UDF code on the server can correctly process it. Little endian format is a format typically used in an Intel and other microprocessor platform.

Updating

(Even if you are not storing the object in little endian format, it is a good idea to prepare the user-supplied attribute data.) Use the DBiPrepareAttrs API to prepare attributes for image objects. Use the DBaPrepareAttrs API to prepare attributes for audio objects. Use 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;
    char hvImgAttrs[50];
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs;
pimgattr->width=640;
pimgattr->height=480;

DBiPrepareAttrs(pimgattr);

EXEC SQL UPDATE EMPLOYEE
    SET PICTURE=REPLACEA(
        PICTURE,
        '/employee/newimg/ajones.bmp',
        :hvStorageType,
        'Anita's new picture',
        :ImgAttrs,                /*user-supplied attributes*/
        'FormatI',
        CAST(NULL as VARCHAR(16384))
    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.

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;
    char hvThumbnail[16384];
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 do not 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;
    char hvVidattrs[3500];
    char hvThumbnail[16384];
EXEC SQL END DECLARE SECTION;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

MMDBVideoAttrs      *pvideoAttr;
pvideoAttr=(MMDBVideoAttrs *)hvVidattrs;

/* Update video content and thumbnail */

EXEC SQL UPDATE EMPLOYEE
    SET VIDEO=REPLACE(
        VIDEO,
        '/employee/newvid/ajones.mpg',
        :hvStorageType,
        'Anita's new video',
        :hvVidAttrs,
        'FormatV',
        'MPEG1',
        :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;
    char hvRemarks [16385];
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 */

strcpy (hvRemarks, "Updated video");

EXEC SQL UPDATE EMPLOYEE
    SET VIDEO=COMMENT(VIDEO, :hvRemarks)
    WHERE NAME = 'Anita Jones';
```

Updating

Use the Replace UDF to update a comment when you update its associated object. For example, the following statements update a video clip that is 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;

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 11. 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 that is stored in a database.

The examples in this chapter demonstrate how to display or play an object from a workstation client. If you want to display or play an object from an OS/390 client, you need to be an OS/390 Open Edition services user. You must also have a browser available that is capable of displaying or playing a multimedia object on the client machine.

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 that is 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

Using display/play APIs

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 409.

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 that are 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 use the value in environment variables to resolve the file's relative file name and path. 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 that are 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 any of the following are true:

- The object is stored as a BLOB
- The relative filename cannot be resolved using the values in environment variables
- 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 in an attribute table 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. The size of the generic video thumbnail that the Video Extender inserts is 108 x 78 pixels. Both the image thumbnail and the generic video thumbnail 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

Displaying thumbnails

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. 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 that is stored in a database table. See “Using the display or play APIs” on page 103 for detailed information on using this API.

Playing an audio or video

Use the DBaPlay API to play an audio that is stored in a database table. Use the DBvPlay API to play a video that is stored in a database table. See “Using the display or play APIs” on page 103 for detailed information on using these APIs.

Chapter 12. Querying images by content

Figure 22 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.

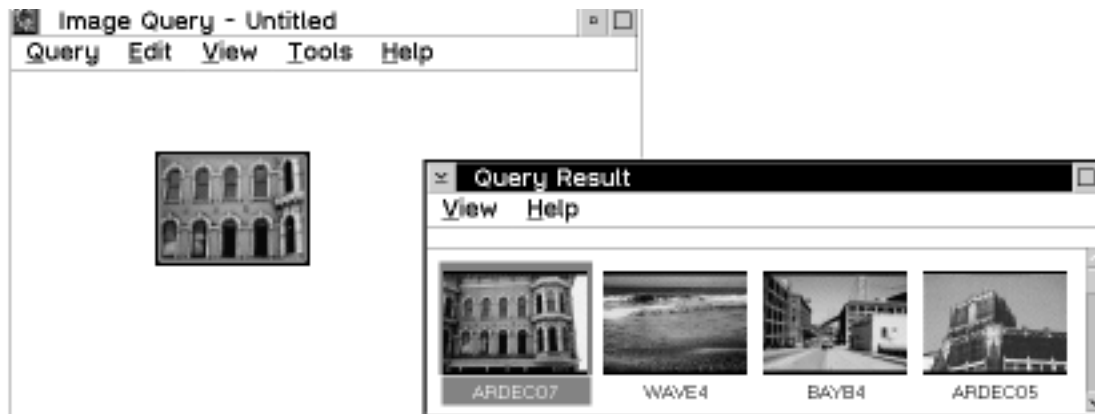


Figure 22. Query by image content. The color or texture of a visual example is used to search for images stored in a database table.

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 that are provided with the Image Extender to build applications like the one just described. It also describes how to use commands and APIs that are 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.
See “QBIC catalogs” on page 16 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 that is called a query string. Alternatively, you can create a query object and associate these attributes with the query object. You can then save the query string and reuse 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

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

How to query by content

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 that are 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 are 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 is a set of administrative support tables that holds data about the visual features of images. The catalog also includes an administrative support table that contains log data.

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. The Image Extender uses an administrative support table to record the association between user table columns and QBIC catalogs.

After you create a QBIC catalog, you can:

- Open the catalog for subsequent actions on it
- Add features to the catalog, this identifies the features for which you want the Image Extender to store data
- Remove features 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
- Catalog images in the catalog
- Recatalog images
- Close the catalog
- Delete the catalog

You can use APIs that are provided by the Image Extender to perform these tasks, including creating a QBIC catalog. You can also perform many of the tasks by 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 need either:

- SYSADM authority
- DBADM authority with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- A user ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; the MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table whose images will be cataloged. The SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

In addition, 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 that you will manually catalog images. Manual cataloging means that you explicitly request the Image Extender to catalog images. (By comparison, the Image Extender for the workstation version of the Image Extender, supports automatic cataloging. Automatic cataloging means the Image Extender automatically catalogs an image after the image is stored in a user table.) See “Manually cataloging a column of images” on page 114 for information on how to manually catalog images.)
- Specify the table spaces and index options for the QBIC catalog tables.

The specification has four parts:

- The name of the table space for the catalog tables that contain feature data. You must specify this table space. The table space should be a segmented table space.
- For the index created on the catalog tables, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. This specification is optional. You get defaults if you do not specify this part.
- The name of the table space for the catalog log table. This specification is optional. The table space for the log table can be a simple table space or a segmented table space. It is recommended that you specify LOCKSIZE PAGE (or accept it as a default) when you create the table space if any of the following are true:
 - Insert or update operations occur infrequently
 - Multiple applications are not inserting, updating, or deleting to or from the same table at the same time
 - Transactions are completed quickly

If you do not specify a table space for the log table, the table space for the feature data tables is used. That table space should be a segmented table space with LOCKSIZE ROW specified.

- For the index created on the log table, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. This specification is optional. You get defaults if you do not specify this part.

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 41 for information on enabling user tables and columns for the Image Extender.)

Using the API: When you use the QbCreateCatalog API, you indicate manual cataloging by specifying a value of 0. In effect, this means automatic cataloging is turned off.

For example, the following statements create a QBIC catalog for the images in the picture column of the employee table. Notice the manual cataloging specification of 0. A table space is specified for the feature data tables in the catalog. Because no

Managing QBIC catalogs

table space is specified for the log table, the table space for the feature table is used. Because no index values are specified, defaults are used.

```
SQLINTEGER autoCatalog=0;                                /* manual cataloging */

rc=QbCreateCatalog(
    "employee",                                           /* user table */
    "picture",                                           /* image column */
    autoCatalog,                                         /* manual cataloging */
    "qbtbspace");                                       /* table space for feature data */
```

Using the command line: When you issue the CREATE QBIC CATALOG command, you indicate manual cataloging by specifying OFF. OFF is the default. You also specify the table spaces for the catalog (and their indexes).

For example, the following command creates the same QBIC catalog as in the API example:

```
CREATE QBIC CATALOG employee picture off USING qbtbspace
```

Back up the QBIC catalog: You should periodically back up the table spaces for the QBIC catalog in case you need to recover the catalog.

Opening a QBIC catalog

You need to open a QBIC catalog to perform subsequent actions that change the catalog. For example, you need to open a QBIC catalog before you add a feature to 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 command OPEN QBIC CATALOG). 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.

Opening the catalog for operations that update it locks the catalog tables in exclusive mode. This can occur immediately or when another API is called. The catalog tables remain locked until the unit of work is committed or rolled back.

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 that are 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.

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	Description
<code>QbColorFeatureClass</code>	Average color
<code>QbColorHistogramFeatureClass</code>	Histogram color
<code>QbDrawFeatureClass</code>	Postional color
<code>QbTextureFeatureClass</code>	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. To include data about a new feature for already cataloged images, you need to recatalog the images (see “Recataloging images” on page 114).

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

Managing QBIC catalogs

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.
- The manual cataloging indicator.

Managing QBIC catalogs

Use the `QbGetCatalogInfo` API to retrieve the user table and column names, the number of features, manual cataloging indicator, and table space specifications. 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;                    /* manual cataloging indicator */
} QbCatalogInfo;
```

When you issue the `QbListFeatures` API call, you need to allocate a buffer to hold the returned feature names. A blank character separates feature names stored in the buffer. 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 that is 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 that is returned by the `QbGetCatalogInfo` API and the `qbiMaxFeatureName` constant is 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, */
/* manual cataloging indicator, and table space specifications */

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
```

Managing QBIC catalogs

Manually cataloging a column of images

After you create a catalog, you manually catalog images that are stored in the user table. When you catalog the images, you catalog an entire column of images. You cannot catalog a single image.

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 newly inserted, updated, or deleted after the column was last cataloged. The Image Extender 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 that is associated with the specified catalog.

For example, the following API call catalogs the uncataloged images in a user table column that is 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”). 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 that is associated with the currently-opened catalog. The images are cataloged for all the features in the catalog:

```
CATALOG QBIC COLUMN FOR NEW
```

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 all the images.

Use the `QbReCatalogColumn` API or the `CATALOG QBIC COLUMN` command to recatalog the images in a QBIC catalog. The Image Extender removes all feature data currently in the catalog. It then analyzes the images for all features, including any new features, and catalogs 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 a column of images” on page 114).

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
```

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 23 on page 117 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 running the complete program, you must run the ENABLE

Managing QBIC catalogs

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 413.

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 */
*****/
#define MMDB_ERROR_MDG_TEXT_LEN 1200

void printError(SQLHSTMT hstmt);
SQLINTEGER createCatalog(void);
SQLINTEGER openCatalog(void);
SQLINTEGER closeCatalog(void);
SQLINTEGER addFeature(void);
SQLINTEGER getCatalogInfo(void);
SQLINTEGER listFeatures(void);
SQLINTEGER catalogImageColumn(void);

QbCatalogHandle cHdl = 0;

SQLHENV henv;
SQLHDBC hdbc;
SQLRETURN rc;
char tableName[] = "sobay_catalog";
char columnName[] = "covers";

SQLCHAR uid[18+1] = "";
SQLCHAR pwd[30+1] = "";
SQLCHAR dbName[SQL_MAX_DSN_LENGTH+1] = "";

int
main(int argc, char *argv[])
{
    if ( ( argc > 4 ) || ( ( argc >= 2 ) && ( strcmp(argv[1], "?") == 0 ) ) )
    {
        printf("Syntax for qbcatdmo \n"
               "    qbcatdmo location_name_or_database userid password\n\n");
        exit(0);
    }
    if (argc > 1)
    {
        strcpy( ( char* ) dbName, argv[1] );
        if (argc > 2) strcpy( ( char* ) uid, argv[2] );
        if (argc > 3) strcpy( ( char* ) pwd, argv[3] );
    }
    else
    {

```

Figure 23. QBIC catalog sample program (Part 1 of 6)

Managing QBIC catalogs

```
/*---- prompt for database name, userid, and password ----*/
printf("Enter database name:\n");
gets((char *) dbName);
printf("Enter userid:\n");
gets((char *) uid);
printf("Enter password:\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);
}
rc = createCatalog();

rc = openCatalog();

if ( rc == MMDB_SUCCESS ) {
    rc = addFeature();

    rc = getCatalogInfo();
    rc = listFeatures();
    rc = catalogImageColumn();
}

rc = closeCatalog();

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

return 0
}
```

Figure 23. QBIC catalog sample program (Part 2 of 6)

```

/*****
/*****
/*****
/*****
SQLINTEGER createCatalog()
{
    SQLINTEGER rc = MMDB_SUCCESS;
    SQLINTEGER autoCatalog = 0;
    char *      tablespaces = NULL;
    SQLINTEGER retLen;
    SQLINTEGER errCode = 0;
    char errMsg[MMDB_ERROR_MSG_TEXT_LEN+1];
    char dbms_name[20] = "";
    SMALLINT dbms_name_sz = 0;

#define SERVER_IS_DB2_390(dbn) ( strcmp(dbn, "DB2" ) ==0 || strcmp( dbn "DSN06010" ) ==0)

    printf("Creating QBIC catalog ...\n");

    /-----*/
    /* find out if we are talking to DB2/390 and if so */
    /* create at least 1 tablespace for the qbic tables */
    /*-----*/

    rc = SQLGetInfo( hdbc, SQL_DBMS_NAME,
        SQLPOINTER) &dbms_name, sizeof(dbms_name), &dbms_name_sz);
    if ( rc != SQL_SUCCESS )
        printError( SQL_NULL_HSTMT );
    else if ( SERVER_IS_DB2_390( dbms_name ) )
    {
        SQLHSTMT hstmt = SQL_NULL_HSTMT;
        char sql_buffer[ MMDB_ERROR_MSG_TEXT_LEN+1 ] = "";
        rc = SQLAllocStmt( hdbc, &hstmt );
        if ( rc != SQL_SUCCESS )
            printError( hstmt );
        else
        {
            sprintf( sql_buffer, "CREATE TABLESPACE SAMPQBIC" ); /* add more if you want */

            rc = SQLExecDirect( hstmt, (SQLCHAR *)sql_buffer, SQL_NTS );
            if ( rc != SQL_SUCCESS )
                printError( hstmt );
            else
                tablespaces = "SAMPQBIC"; /* set 1 tablespace for qbic tables */

            SQLFreeStmt(hstmt, SQL_DROP);
        }
    }
}

```

Figure 23. QBIC catalog sample program (Part 3 of 6)

Managing QBIC catalogs

```
/*-----*/
/* now create the qbic catalog */
/*-----*/
if ( rc == MMDB_SUCCESS )
{
    rc = QbCreateCatalog( 3
                        (char *) tableName,
                        (char *) columnName,
                        autoCatalog,
                        tablespaces
                        );
    if ( rc != MMDB_SUCCESS ) {
        DBiGetError(&errCode, errMsg);
        printf("Error code is %d Error Message %s", errCode, errMsg);
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
    }
}
else
    SQLTransact(henv, hdbc, SQL_COMMIT);
}

/*****
****/
**** openCatalog():
****/
****/
****/
SQLINTEGER openCatalog()
{
    SQLINTEGER rc = MMDB_SUCCESS
    SQLINTEGER errCode = 0;
    char errMsg[MMDB_ERROR_MSG_TEXT_LEN+1];
    SQLINTEGER mode = qbiUpdate;

    printf("Opening QBIC catalog...\n");
    rc=QbOpenCatalog( 4
                    (char *) tableName,
                    (char *) columnName,
                    mode,
                    &cHdl
                    );

    if ( rc != MMDB_SUCCESS ) {
        DBiGetError(&errCode, errMsg);
        printf("Error code is %d Error Message %s", errCode, errMsg);
        SQLTRANSACT(henv, hdbc, SQL_ROLLBACK);
    }
}
```

Figure 23. QBIC catalog sample program (Part 4 of 6)


```

/*****
/**** addFeature() ****
/****
/****
*****/
SQLINTEGER addFeature()
{
    SQLINTEGER rc = MMDB_SUCCESS;
    SQLINTEGER errCode = 0;
    char errMsg[MMDB_ERROR_MSG_TEXT_LEN+1];

    char CfeatureName[] = "QbColorFeatureClass";
    char DfeatureName[] = "QbDrawFeatureClass";
    char CHfeatureName[] = "QbColorHistogramFeatureClass";
    char TfeatureName[] = "QbTextureFeatureClass";

    printf("Adding Features Class ... \n");
    if(cHdl) /* if we have an open catalog, else do nothing */
    {
        printf(" Color Feature Class ... &n");
        rc = QbAddFeature(
            cHdl,
            CfeatureName 5
        );
        if(rc!=MMDB_SUCCESS){
            DBiGetError(&errCode, errMsg);
            print("Error code is %d Error Message %s\n",errCode, errMsg);
            SQLTransact(henv, hdbc, SQL_ROLLBACK);
        }
        else
            SQLTransact(henv, hdbc, SQL_COMMIT);
    }
    rc=16;

    return rc ;
}

/*****
/****
/**** catalogImageColumn() ****
/****
*****/
SQLINTEGER catalogImageColumn()
{
    SQLINTEGER rc = MMDB_SUCCESS;
    SQLINTEGER errCode = 0;
    char errMsg[MMDB_ERROR_MSG_TEXT_LEN+1];

    printf("Cataloging image column. Please wait ... \n");
    if ( cHdl ) /* if we have an open catalog else do nothing */
    {
        SQLRETURN rc;
    }

```

Figure 23. QBIC catalog sample program (Part 5 of 6)

Building queries

```
rc=QbCatalogColumn( 6
if ( rc != MMDB_SUCCESS ) {
DBiGetError(&errCode, errMsg);
printf("Error code is %d Error Message %s\n",errCode, errMsg);
if( errCode !=qbicECcatalogingErrors )
    SQLTransact(henv, hdbc, SQL_ROLLBACK);
}
else
SQLTransact(henv, hdbc, SQL_COMMIT);
}
else
rc = 16 ;

return rc ;
}
/*****
*****/
closeCatalog()
*****/
SQLINTEGER closeCatalog()
{
SQLINTEGER rc = MMDB_SUCCESS;
SQLINTEGER errCode = 0;
char errMsg[MMDB_ERROR_MSG_TEXT_LEN+1];

printf("Closing QBIC catalog ...\n");
if ( cHdl ) /* if we have an open catalog close it else do nothing */
{
rc=QbCloseCatalog( 7
if ( rc != MMDB_SUCCESS ) {
DBiGetError(&errCode, errMsg);
printf("Error code is %d Error Message %s\n",errCode, errMsg);
SQLTransact(henv, hdbc, SQL_ROLLBACK);
}
}
else
rc = 16 ;

return rc ;
}
/*****/
```

Figure 23. QBIC catalog sample program (Part 6 of 6)

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 refer to 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”. 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 124. 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 131 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. You must enclose file names that contain spaces or closing angle brackets (>) in double quotation marks; other file names can optionally be enclosed in double quotation marks. If you use quotation marks surrounding a file name, each quotation mark 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.

In the following example, a query string is passed within a DB2 command:

```
db2 "select image_id from table
(mmdbsys.QbScoreTBFromStr
('texture 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
averageColor, average, or QbColorFeatureClass	<p>color=<Rvalue, Gvalue, Bvalue></p> <p>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.</p> <p>file=<file_location, filename></p> <p>The <i>file_location</i> is server for a server file. The <i>filename</i> is the complete file path specified in the format appropriate for the system in which the file resides, or a relative file name. DB2 extenders resolves the relative file name using environment variables (see “How environment variables are used to resolve file names” on page 409).</p>

Building queries

Table 9. Feature values that can be specified in query string (continued)

Feature name	Value
histogram, histogramcolor, or QbColorHistogramFeatureClass	<p>histogram=<(hist_value, Rvalue, Gvalue, Bvalue)>, ...</p> <p>Each histogram color value is specified in a clause that identifies the percent (1 to 100) of that color in the histogram (<i>hist_value</i>), and the red value (<i>Rvalue</i>), green value (<i>Gvalue</i>), and blue value (<i>Bvalue</i>) of that color.</p> <p>file=<file_location, filename></p> <p>The <i>file_location</i> is server for a server file. The <i>filename</i> is the complete file path specified in the format appropriate for the system in which the file resides, or a relative file name. DB2 extenders resolves the relative file name using environment variables.</p>
draw, positional, or QbDrawFeatureClass	<p>file=<file_location, filename></p> <p>handle=<image_handle></p> <p>The <i>file_location</i> is server for a server file. The <i>filename</i> is the complete file path specified in the format appropriate for the system in which the file resides, or a relative file name. DB2 extenders resolves the relative file name using environment variables.</p>
texture or QbTextureFeatureClass	<p>file=<file_location, filename></p> <p>handle=<image_handle></p> <p>The <i>file_location</i> is server for a server file. The <i>filename</i> is the complete file path specified in the format appropriate for the system in which the file resides, or a relative file name. DB2 extenders resolves the relative file name using environment variables.</p>

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 do not 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 that is 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 that are 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:

```
averageColor color=<255, 0, 0>
```

The following query string specifies a histogram comprised of 10% red, 50% green, and 40% blue:

```

| histogram 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:

```

| averageColor color=<30, 200, 25> and
| texture 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.

After you create a query object, you can optionally name and save it (see “Saving and reusing a query string” on page 129). However before you create the first named query object, you need to ensure that the job DMBSETUP is properly edited and run. The job creates an administrative support table that the Image Extender uses to record information about named queries. For further information, see “Editing and running job DMBSETUP”.

The Image Extender provides three APIs (QbQuerySearch, QbQueryStringSearch, 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 131 for details.)

Editing and running job DMBSETUP

Provided with the DB2 extenders is a job named DMBSETUP. A system administrator needs to run the job to initialize the DB2 extenders after the extenders are installed. (Installing and initializing the DB2 extenders is described in *Program Directory for IBM Database 2 Universal Database Server for OS/390 Volume 1 of 8*.) Within the DMBSETUP job there are SQL statements that create an administrative support table for recording information about named queries. The job also includes SQL statements that create the table space for the table and create an index for the table. The SQL statements appear as comments, so they have to be uncommented before they are run. The statements are as follows:

```

CREATE TABLESPACE QBICNQTS IN MMDBSYS
  USING STOGROUP SYSDEFLT PRIQTY 12 SEQTY 12 LOCKSIZE ROW;

CREATE TABLE MMDBSYS.QBICQUERIES( NAME CHAR(18) NOT NULL,
                                   DESCRIPTION CHAR(250),
                                   QUERY VARCHAR(1024))
  IN MMDBSYS.QBICNQTS;

CREATE UNIQUE INDEX MMDBSYS.QBICQUERIESX ON
  MMDBSYS.QBICQUERIES( NAME );

```

Do not change the MMDBSYS database specification in the CREATE TABLESPACE statement. Similarly, do not change the name of the table and the column specifications in the CREATE TABLE statement, or the table name in the CREATE UNIQUE INDEX statement. However you can edit other specifications in the

Building queries

statements as appropriate. After editing the job, you need to ensure that a system administrator runs it before you create the first named query object.

Appropriate privilege needs to be granted on the table that records information about named queries. SELECT privilege on the table is required to refer to the name in a query. INSERT and DELETE privilege on the table is required to create and delete named queries.

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.

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 object handle.
- Name the feature.
- Point to the QbImageSource structure (see page 127 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 reserved;
        QbImageBuffer reserved;
        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_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.

If you set the type field to qbiSource_ServerFile, use clientFile for the name and type of the file on the server.

Depending on the type of source, the Image Extender also examines other information that you specify. This is shown in Table 10.

Table 10. What the Image Extender examines in QbImageSource

Source	What the Image Extender examines	Where specified
a user table	image handle	image handle field of QbImageSource
average color specification	red, green, and blue color values	QbColor structure (see page 127 for details about using this structure)
histogram color specification	color values and percentages	QbHistogramColor structure (see page 128 for details about using this structure)

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:

Building queries

```
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 that is 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 that is 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 an image in a user table.

```
char          featureName[qbiMaxFeatureName];
QbQueryHandle qHandle;
QbImageSource imgSource;

imgSource.type=qbiSource_ImageHandle;
strcpy(imgSource.imageHandle,handle);

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 do not 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 that is 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 */
```

Saving and reusing a query string

Query objects are transient unless you save them. They exist only when the application runs. Saving a query string allows you to use that query string again in your program or across program invocations.

The Image Extender provides the `QbQueryGetString` API that returns the query string from a query object. You can then use that query string as input to the `QbQueryStringSearch` API or to the `QbScoreFromStr` and `QbScoreTBFromStr` UDFs in other queries by image content (see “Issuing queries by image content” on page 131).

The query string is built when you build the query using:

- `QbQueryCreate`
- `QbQueryAddFeature`
- `QbQuerySetFeatureData`
- `QbQuerySetFeatureWeight`
- `QbQueryRemoveFeature`

Building queries

After you build the query, you can call `QbQueryGetString` to get the string. You can use this query string in calls within that program or save it to a file for use in subsequent invocations of your application and in other database connections. After you are finished using the query string returned by `QbQueryGetString`, you must explicitly free the space.

In the following example, the `QbQueryGetString` is used to retrieve the query string from a query object:

```
SQLRETURN rc;
char* qryString;
QbQueryHandle qHandle;

.....          /* Here you create and use the query */

rc = QbQueryGetString(qHandle, &qryString);
if ( rc == 0) {
    ...          /* Use the query string as input here */
    free((void *)qryString);
    qryString=(char *)0;
}
```

Restriction:: When you use a client file to specify the data source for a feature, the query string does not reflect the feature data.

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
<code>QbQueryGetFeatureCount</code>	The number of features in a query object
<code>QbQueryListFeatures</code>	The names of the features 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 */
```

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 an unnamed query object with the `QbQueryDelete` API. The Image Extender deletes the query from the currently connected database.

When you use the `QbQueryDelete` API, specify the query object handle.

In the following example, the `QbQueryDelete` API is used to delete a query object:

```
QbQueryHandle qHandle;

rc=QbQueryDelete(
    qHandle);          /* query object handle */
```

If you have used a named query, delete the query object with the `QbQueryNameDelete` API.

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 123) or in a query object (see “Using a query object” on page 125).

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, you submit the query from within a program, by referencing its handle.

The Image Extender compares the feature values that are 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 that is 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.

Issuing QBIC queries

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 that uses 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 */
```

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

Recommendation: Use the QbScoreFromStr UDF to get the score of a single cataloged image. Use the QbScoreTBFromStr UDF to get the score of multiple cataloged images in a table column.

Retrieving the score of a single image

Use the QbScoreFromStr UDF to get the score of a single cataloged image in a table column. Specify a query string as input to the QbScoreFromStr UDF. If you use the QbScoreFromName 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.

To use these UDFs, you need:

- SELECT privilege on the MMDBSYS.QBICQUERIES table
- SELECT privilege on the QBIC Catalog tables
- SELECT privilege on administrative support tables, for any handles specified in the query
- Authority to reference any files specified in the query

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
decimal (QbScoreFromStr(swatch_img,
    'QbColorFeatureClass color=<255, 0, 0>'), /* query string *
```

Issuing QBIC queries

```
10, 5) AS score
FROM fabric
ORDER BY score
```

/* table column */

Retrieving the score of multiple images

Use the QbScoreTBFromStr UDF to get the score of multiple cataloged images in a table column. You can use the QbScoreTBFromName UDF, if you have a named query. Both UDFs return a two-column table of image handles and scores; the rows in the table are in ascending order of score. The name of the handle column in the result table is IMAGE_ID; the name of the score column is SCORE.

To use these UDFs, you need:

- SELECT privilege on the MMDBSYS.QBICQUERIES table
- SELECT privilege on the QBIC Catalog tables
- SELECT privilege on administrative support tables, for any handles specified in the query
- Authority to reference any files specified in the query

Specify a query string as input to the QbScoreTBFromStr 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));
```

QBIC query sample program

Figure 24 on page 136 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 that is 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 run 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 413.

Note the following points in Figure 24 on page 136:

- 1** Include the `dmbqbapi` header file.
- 2** Prompt the user for database information.
- 3** Connect to the database.
- 4** Create a query object.
- 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 106.
- 10** Delete the query object.

The `SAMPLES` subdirectory includes another program that demonstrates how to build and use a QBIC query. The program, `QbicQry.java`, shows you how 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.

Issuing QBIC queries

```
#include <sql.h>
#include <sqlcli.h>
#include <sqlcli1.h>
#include <dmbqbqpi.h> 1
#include <stdio.h>
#include <string.h>
#ifdef DMB_MVS
#include <stdlib.h>
#else
#include <malloc.h>
#endif
#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* tableName;
static char* columnName;

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 24. QBIC query sample program (Part 1 of 7)


```

/*
123456789012345678901234567890123456789012345678901234567890 */
"" ,
"+-----+",
" AVERAGE COLOR colorname      |",
" AVERAGE FILE filename format |",
" AVERAGE LAST                  |",
" HISTOGRAM COLOR number colorname [ number colorname ... ] |",
" HISTOGRAM FILE filename format |",
" HISTOGRAM 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 commands may be abbreviated to their first letter except string & help.",
"" ,
">>pause<<",
0 ,
};

```

Figure 24. QBIC query sample program (Part 2 of 7)

Issuing QBIC queries

```

/*****
/* doBrowse()
/*
/*
/*
/*****/
static void doBrowse(char* handle)
{
    static char msg[1000];
    SQLINTEGER sqlcode;
    int ret;

    ret = DBiBrowse(0, MMDB_PLAY_HANDLE, handle, MMDB_PLAY_NO_WAIT); 9
    if (ret == -818) {
        printf("Database not correctly bound to the image extender browser.\n");
    }
    else if (ret != 0) {
        DBiGetError(&sqlcode, msg);
        printf("Error from DBiBrowse():\n");
        printf("\tReturn code:  %d\n", ret);
        printf("\tSqlcode:      %05.5d\n", sqlcode);
        printf("\tMessage:      '%s'\n", msg);
    }
}

/*****
/* 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;
        }
    }
}
```

Figure 24. QBIC query sample program (Part 3 of 7)

```

    }
    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;
        }
    }
    else {
        printf("AVERAGE command only supports COLOR, FILE, and LAST types.\n");
        return;
    }

    _QbQuerySetFeatureData(averageHandle, "QbColorFeatureClass", &is); 7
    _QbQuerySearch(averageHandle, tableName, columnName, MaxQueryReturns,
        0, 0, &imageCount, results); 8
    lastHandle = averageHandle;

    currentImage = -1;
}

```

Figure 24. QBIC query sample program (Part 4 of 7)

Issuing QBIC queries

```

/*****
/* 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("> ");
        fflush(stdout);
        gets(line);
        done = processCommand(line);
    }
}

```

Figure 24. QBIC query sample program (Part 5 of 7)

```

/*****
/* main()
/*
/*
/*
/*****
int
main(int argc, char *argv[]) {
    char* inst;
    int i;

#ifdef DMB_HP
    _main();
#endif

    if ( ( argc > 4 ) || ( ( argc >= 2 ) && ( strcmp(argv[1], "?") == 0 ) ) )
    {
        printf( "Syntax for qbicdemo \n"
            "    qbicdemo location_name_or_database userid password\n\n" );
        exit(0);
    }

    if (argc > 1)
    {
        strcpy( (char *) databaseName, argv[1] );
        if (argc > 2) strcpy( (char *) userid, argv[2] );
        if (argc > 3) strcpy( (char *) password, argv[3] );
    }
    else
    {
        /*---- prompt for database name, userid, and password ----*/
        printf("Enter database name:\n"); 2
        gets((char *) databaseName);
        printf("Enter userid:\n");
        gets((char *) userid);
        printf("Enter password:\n");
        gets((char *) 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);

```

Figure 24. QBIC query sample program (Part 6 of 7)

Issuing QBIC queries

```
printf("Initializing . . .\n");
tableName = "SOBAY_CATALOG";
columnName = "COVERS";

printf("Create query object ...\n");
_QbQueryCreate(&averageHandle);

printf("Adding 'Average Color' feature to Query object ...\n");
_QbQueryAddFeature(averageHandle, "QbColorFeatureClass");

printf("Create query object ...\n");
_QbQueryCreate(&histogramHandle);

printf("Adding 'Histogram Color' feature to Query object ...\n");
_QbQueryAddFeature(histogramHandle, "QbColorHistogramFeatureClass");

printf("Create query object ...\n");
_QbQueryCreate(&drawHandle); 4

printf("Adding 'Draw' feature to Query object ...\n");
_QbQueryAddFeature(drawHandle, "QbDrawFeatureClass"); 5
printf("Starting query commands ...\n");
commandLoop();

printf("Deleting query objects ...\n");
_QbQueryDelete(drawHandle); 10
_QbQueryDelete(histogramHandle);
_QbQueryDelete(averageHandle);

printf("Freeing resources ...\n");
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);

printf("\nDone!\n");
}
```

Figure 24. QBIC query sample program (Part 7 of 7)

Part 4. Reference

Chapter 13. User-defined types (distinct types) and user-defined functions

Schema	145
User-defined types (distinct types)	145
User-defined functions	145
AlignValue	149
AspectRatio	150
BitsPerSample	151
BytesPerSec	152
Comment	153
CompressType	155
Content	156
ContentA	160
DB2Audio	162
DB2AudioA	164
DB2Image	167
DB2ImageA	170
DB2Video	172
DB2VideoA	174
Duration	176
Filename	177
FindInstrument	178
FindTrackName	179
Format	180
FrameRate	181
GetInstruments	182
GetTrackNames	183
Height	184
Importer	185
ImportTime	186
MaxBytesPerSec	187
NumAudioTracks	188
NumChannels	189
NumColors	190
NumFrames	191
NumVideoTracks	192
QbScoreFromName	193
QbScoreFromStr	195
QbScoreTBFromName	196
QbScoreTBFromStr	198
Replace	200
ReplaceA	203
SamplingRate	206
Size	207
Thumbnail	208
TicksPerQNote	210
TicksPerSec	211
Updater	212
UpdateTime	213
Width	214

Chapter 14. Application programming interfaces

DBaAdminGetInaccessibleFiles	216
DBaAdminGetReferencedFiles	218
DBaAdminIsFileReferenced	220
DBaDisableColumn	222

DBaDisableServer	224
DBaDisableTable	225
DBaEnableColumn	227
DBaEnableServer	229
DBaEnableTable	231
DBaGetError	233
DBaGetInaccessibleFiles	234
DBaGetReferencedFiles	236
DBaIsColumnEnabled	238
DBaIsFileReferenced	240
DBaIsServerEnabled	242
DBaIsTableEnabled	243
DBaPlay	245
DBaPrepareAttrs	247
DBiAdminGetInaccessibleFiles	248
DBiAdminGetReferencedFiles	250
DBiAdminIsFileReferenced	252
DBiBrowse	254
DBiDisableColumn	256
DBiDisableServer	258
DBiDisableTable	259
DBiEnableColumn	261
DBiEnableServer	263
DBiEnableTable	265
DBiGetError	267
DBiGetInaccessibleFiles	268
DBiGetReferencedFiles	270
DBiIsColumnEnabled	272
DBiIsFileReferenced	274
DBiIsServerEnabled	276
DBiIsTableEnabled	277
DBiPrepareAttrs	278
DBvAdminGetInaccessibleFiles	279
DBvAdminGetReferencedFiles	281
DBvAdminIsFileReferenced	283
DBvDisableColumn	285
DBvDisableServer	287
DBvDisableTable	288
DBvEnableColumn	290
DBvEnableServer	292
DBvEnableTable	294
DBvGetError	296
DBvGetInaccessibleFiles	297
DBvGetReferencedFiles	299
DBvIsColumnEnabled	301
DBvIsFileReferenced	303
DBvIsServerEnabled	305
DBvIsTableEnabled	306
DBvPlay	307
DBvPrepareAttrs	309
QbAddFeature	310
QbCatalogColumn	312
QbCloseCatalog	314
QbCreateCatalog	315
QbDeleteCatalog	317
QbGetCatalogInfo	319

QbListFeatures	320
QbOpenCatalog	322
QbQueryAddFeature	324
QbQueryCreate.	326
QbQueryDelete.	327
QbQueryGetFeatureCount	328
QbQueryGetString.	329
QbQueryListFeatures	330
QbQueryNameCreate.	332
QbQueryNameDelete.	334
QbQueryNameSearch	335
QbQueryRemoveFeature	337
QbQuerySearch.	339
QbQuerySetFeatureData	341
QbQuerySetFeatureWeight	343
QbQueryStringSearch.	344
QbReCatalogColumn	346
QbRemoveFeature.	348

Chapter 15. Administration commands for the client

Entering DB2 extender administration commands	351
Getting online help for DB2 extender commands	352
ADD QBIC FEATURE	353
CATALOG QBIC COLUMN	354
CLOSE QBIC CATALOG	355
CREATE QBIC CATALOG	356
DELETE QBIC CATALOG	358
DISABLE COLUMN	359
DISABLE SERVER.	360
DISABLE TABLE	361
ENABLE COLUMN	362
ENABLE SERVER.	363
ENABLE TABLE	365
GET EXTENDER STATUS	367
GET INACCESSIBLE FILES	368
GET QBIC CATALOG INFO	370
GET REFERENCED FILES	371
GRANT	373
OPEN QBIC CATALOG	375
QUIT	376
REMOVE QBIC FEATURE	377
REVOKE	378
TERMINATE	380

Chapter 16. Diagnostic information

Handling UDF return codes	381
Handling API return codes	382
SQLSTATE codes	382
Messages	386
Diagnostic tracing	404
Start tracing	404
Stop tracing	405
Reformat trace information.	405

Chapter 13. User-defined types (distinct types) and user-defined functions

This chapter gives reference information for the UDTs and UDFs created by the DB2 extenders.

Schema

The extenders use the MMDBSYS schema for all of their object-relational objects, including UDTs and UDFs.

User-defined types (distinct types)

Table 11 lists and describes the user-defined types created by the DB2 extenders. It also lists the DB2 source data type for each UDT.

Table 11. User-defined types created by the DB2 extenders

UDT	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 provide 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 that is returned by the UDF, including its data type
- Examples of use

User-defined functions

Table 12 lists the UDFs and identifies the extenders that provide 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 12. 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	149
AspectRatio	Returns the aspect ratio of the first track of an MPEG1 and MPEG2 video.			x	150
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	151
BytesPerSec	Returns the data transfer rate, in average bytes per second, for a WAVE audio.		x		152
Comment	Returns or updates a comment stored with an image, audio, or video.	x	x	x	153
CompressType	Returns the compression format, such as MPEG-1, of a video.			x	155
Content	Retrieves or updates the content of an image, audio, or video from a database.	x	x	x	156
ContentA	updates the content of an image, audio, or video with user-supplied attributes from a database.	x	x	x	160
DB2Audio	Stores the content of an audio in a database table.		x		162
DB2AudioA	Stores the content of an audio with user-supplied attributes in a database table.		x		164
DB2Image	Stores the content of an image in a database table.	x			167
DB2ImageA	Stores the content of an image with user-supplied attributes in a database table.	x			170
DB2Video	Stores the content of a video in a database table.			x	172
DB2VideoA	Stores the content of a video with user-supplied attributes in a database table.			x	174
Duration	Returns the duration (that is, playing time in seconds) of a WAVE or AIFF audio, or video.		x	x	176
Filename	Returns the name of the server file that contains the contents of an image, audio, or video.	x	x	x	177
FindInstrument	Returns the track number of the first occurrence of a specified instrument in a MIDI audio.		x		178

Table 12. DB2 Extender UDFs (continued)

UDF	Description	Image	Audio	Video	See page
FindTrackName	Returns the number of a specified named track in a MIDI audio.		x		179
Format	Returns the format of an image, audio, or video.	x	x	x	180
FrameRate	Returns the throughput of a video in frames per second.			x	181
GetInstruments	Returns the instrument name of all instruments in a MIDI audio.		x		182
GetTrackNames	Returns the name of all tracks in a MIDI audio.		x		183
Height	Returns the height, in pixels, of an image or video frame.	x		x	184
Importer	Returns the user ID of the person who stored an image, audio, or video in a database table.	x	x	x	185
ImportTime	Returns a timestamp that indicates when an image, audio, or video was stored in a database table.	x	x	x	186
MaxBytesPerSec	Returns the maximum throughput of a video in bytes per second.			x	187
NumAudioTracks	Returns the number of audio tracks in a video or MIDI audio.		x	x	188
NumChannels	Returns the number of recorded audio channels in a WAVE or AIFF audio, or video.		x	x	189
NumColors	Returns the number of colors in an image.	x			190
NumFrames	Returns the number of frames in a video.			x	191
NumVideoTracks	Returns the number of video tracks in a video.			x	192
QbScoreFromName	Returns the score of an image (uses a named query object). (Replaces QbScore.)	x			193
QbScoreFromStr	Returns the score of an image (uses a query string).	x			195
QbScoreTBFromName	Returns a table of scores from an image column (uses a named query object).	x			196
QbScoreTBFromStr	Returns a table of scores from an image column (uses a query string).	x			198
Replace	Updates the content of an image, audio, or video stored in a database, and updates its comment.	x	x	x	200
ReplaceA	Updates the content of an image, audio, or video with user-supplied attributes stored in a database, and updates its comment.	x	x	x	203

User-defined functions

Table 12. DB2 Extender UDFs (continued)

UDF	Description	Image	Audio	Video	See page
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	206
Size	Returns the size of an image, audio, or video, in bytes.	x	x	x	207
Thumbnail	Returns or updates a thumbnail-size version of an image or video frame stored in a database.	x		x	208
TicksPerQNote	Returns the clock speed of a recorded MIDI audio, in ticks per quarter note.		x		210
TicksPerSec	Returns the clock speed of a recorded MIDI audio, in ticks per second.		x		211
Updater	Returns the user ID of the person who last updated an image, audio, or video in a database table.	x	x	x	212
UpdateTime	Returns a timestamp that indicates when an image, audio, or video in a database table was last updated.	x	x	x	213
Width	Returns the width in pixels of an image or video frame.	x		x	214

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

►►—AlignValue—(*—handle—*)—————►

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 that are stored in the sound column of the employee table that are word aligned:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvAud_fname[255];
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

AspectRatio

Image	Audio	Video
		X

Returns the aspect ratio of the first track of an MPEG video.

Include file

dmbvideo.h

Syntax

►► AspectRatio(*--handle--*)◄◄

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 that is 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

►► BitsPerSample (—*handle*—) ◀◀

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[255];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
INTO :hvAud_fname
FROM EMPLOYEE
WHERE FORMAT(SOUND)='WAVE'
AND BITSPERSAMPLE(SOUND) = 8;
```

BytesPerSec

BytesPerSec

Image	Audio	Video
	X	

Returns the data transfer rate, in average bytes per second, for a WAVE audio.

Include file

dmbaudio.h

Syntax

►► BytesPerSec(—*handle*—) ◄◄

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[255];
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 that is stored with an image, audio, or video.

Include file

image dmbimage.h
audio dmbaudio.h
video dmbvideo.h

Syntax

Retrieve comment

►►—Comment—(—*handle*—)—————►►

Syntax

Update comment

►►—Comment—(—*handle*—,—*new_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. An 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 that is associated with the Anita Jones’s video clip in the video column of the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvRemarks[4000];
EXEC SQL END DECLARE SECTION;
```

Comment

```
/* Get the old comment */

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

/* Update the comment */

strcpy (hvRemarks, "Updated video");

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

►►—CompressType—(*—handle—*)—————►◄

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 that are stored in the video column of the employee table whose compression format is MPEG-1:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvVid_fname[255];
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, workstation client file, or server file.

Include file

image dmbimage.h
audio dmbaudio.h
video dmbvideo.h

Syntax

Retrieve content to buffer or workstation client file

►► Content—(*—handle—*)—————►◄

Syntax

Retrieve a segment of content to buffer or workstation client file

►► Content—(*—handle—, —offset—, —size—*)—————►◄

Syntax

Retrieve content to server file

►► Content—(*—handle—, —target_file—, —overwrite—*)—————►◄

Syntax

Retrieve content to buffer or workstation client file with format conversion—image only

►► Content—(*—handle—, —target_format—*)—————►◄

Syntax

Retrieve content to server file with format conversion—image only

►► Content—(*—handle—, —target_file—, —overwrite—, —target_format—*)—————►◄

Syntax

Retrieve content to buffer or workstation client file with format conversion and additional changes—image only

►► Content—(*—handle—, —target_format—, —conversion_options—*)—————►◄

Syntax

Retrieve content to server file with format conversion and additional changes—image only

```
►►Content—(—handle—,—target_file—,—overwrite—,——————→
►—target_format—,—conversion_options—)—————→◄◄
```

Syntax

Update content from buffer or workstation client file

```
►►Content—(—handle—,—content—,—source_format—,—target_file—)—————→◄◄
```

Syntax

Update content from server file

```
►►Content—(—handle—,—source_file—,—source_format—,—stortype—)—————→◄◄
```

Syntax

Update content from buffer or workstation client file with format conversion—image only

```
►►Content—(—handle—,—content—,—source_format—,——————→
►—target_format—,—target_file—)—————→◄◄
```

Syntax

Update content from server file with format conversion—image only

```
►►Content—(—handle—,—source_file—,—source_format—,——————→
►—target_format—,—target_file—)—————→◄◄
```

Syntax

Update content from buffer or workstation client file with format conversion and additional changes—image only

```
►►Content—(—handle—,—content—,—source_format—,——————→
►—target_format—,—conversion_options—,—target_file—)—————→◄◄
```

Syntax

Update content from server file with format conversion and additional changes—image only

```
►Content—(—handle—,—source_file—,—source_format—,——————►
►target_format—,—conversion_options—,—target_file—)—————►
```

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 (VARCHAR(254))

The name of the file that contains the content for update of the image, audio, or video.

target_file (VARCHAR(254))

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 72 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.

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 that is stored for Anita Jones in the picture column of the employee table:

```
char hvImg_fname[255];
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';
```

ContentA

Image	Audio	Video
X	X	X

Updates the content of an image, audio, or video with user-supplied attributes from a database. The content can be retrieved to a client buffer, workstation client file, or server file.

Include file

image dmbimage.h

audio dmbaudio.h

video dmbvideo.h

Syntax

Update content with user-supplied attributes from buffer or workstation client file

```

▶▶ContentA—(—handle—,—content—,——————→
▶—target_file—,—attrs—,—format—,—compress_type—,—thumbnail—)—————<<

```

Syntax

Update content with user-supplied attributes from server file

```

▶▶ContentA—(—handle—,—source_file—,—stortype—,—attrs—,——————→
▶—format—,—compress_type—,—thumbnail—)—————<<

```

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 (VARCHAR(254))

The name of the file that contains the content for update of the image, audio, or video.

target_file (VARCHAR(254))

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.

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.

format (VARCHAR(8))

The format of the source for update of the image, audio, or video.

attrs (VARCHAR (4096) FOR BIT DATA)

The attributes of the image, audio, or video

compress_type (VARCHAR (8))

The compression format of the video (video only)

thumbnail (VARCHAR (16384) FOR BIT DATA)

A thumbnail of the image or video frame (image and video only)

Return values (data type)

The handle of the image, audio, or video to be updated (DB2IMAGE, DB2AUDIO, or DB2VIDEO).

Examples

Update the image that is stored for Anita Jones in the picture column of 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.

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
char hvImgattrs[100];
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs;
pimgattr->width=640;
pimgattr->height=480;

DBiPrepareAttrs(pimgattr);

EXEC SQL UPDATE EMPLOYEE
  SET VIDEO=CONTENTA(
    PICTURE,
    '/employee/newimg/ajones.bmp',
    :hvStorageType,
    :ImgAttrs,
    'FormatI',
    '')
  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, workstation client file, or server file. The audio can be stored in the database table as a BLOB, or in a server file (referred to 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 workstation client file

```
►►DB2Audio—(—dbname—,—content—,—format—,—target_file—,—comment—)——►◄
```

Syntax

Store content from server file

```
►►DB2Audio—(—dbname—,—source_file—,—format—,—stortype—,—comment—)——►◄
```

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 71 for supported audio formats.

target_file (VARCHAR(254))

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 (VARCHAR(254))

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 VARCHAR(16384))

A comment to be stored with 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',
        '',
        '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'));
```

DB2AudioA

Image	Audio	Video
	X	

Stores the content of an audio with user-supplied attributes in a database table. The audio source can be in a client buffer, workstation client file, or server file. The audio can be stored in the database table as a BLOB, or in a server file (referred to 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 with user-supplied attributes from buffer or workstation client file

```

>> DB2AudioA(—dbname—,—content—,—target_file—,——————→
           ►comment—,—attrs—,—tracknames—,—instruments—,—format—)—————→<<

```

Syntax

Store content with user-supplied attributes from server file

```

>> DB2AudioA(—dbname—,—source_file—,—stortype—,—comment—,——————→
           ►attrs—,—tracknames—,—instruments—,—format—)—————→<<

```

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 71 for supported audio formats.

target_file (VARCHAR(254))

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 (VARCHAR(254))

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 (VARCHAR(16384))

A comment to be stored with the audio.

attrs (VARCHAR(4096) FOR BIT DATA)

The attributes of the audio.

tracknames (VARCHAR(1536))

The names of all tracks in the MIDI audio. The values are in track number order (for example, PIANO TUNE; TRUMPET FANFARE). If a track does not have a name, its field is blank. A null value should be specified for audio formats other than MIDI.

instruments (VARCHAR(1536))

The names of all instruments in the MIDI audio. The values are in track number order (for example, PIANO; TRUMPET; BASS). If a track does not have an associated instrument, its field is blank. A null value should be specified for audio formats other than MIDI.

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. 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;
char hvAudattr[600];
EXEC SQL END DECLARE SECTION;

MMDBAudioAttrs      *paudiattr;

hvStorageType = MMDB_STORAGE_TYPE_INTERNAL;

paudioattr=(MMDBAudioAttrs *) hvAudattr;
paudioAttr->ulSamplingRate=44100;
paudioAttr->usNumChannels=2;

EXEC SQL INSERT INTO EMPLOYEE VALUES(
    '128557',
    'Anita Jones',
    DB2AUDIOA(
        CURRENT SERVER,
        '/employee/sounds/ajones.aud',
        :hvStorageType,
        'Anita"s voice',
        :hvAudattr,
```

DB2AudioA

```
    '' ,  
    '' ,  
    'FormatA')  
);
```

DB2Image

Image	Audio	Video
X		

Stores the content of an image in a database table. The image source can be in a client buffer, workstation client file, or server file. The image can be stored in the database table as a BLOB, or in a server file (referred to 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 workstation client file

```

>>> DB2Image—(—dbname—,—content—,—source_format—,——————>
>—target_file—,—comment—)——————><

```

Syntax

Store content from server file

```

>>> DB2Image—(—dbname—,—source_file—,—source_format—,——————>
>—stortype—,—comment—)——————><

```

Syntax

Store content from buffer or workstation client file with format conversion

```

>>> DB2Image—(—dbname—,—content—,—source_format—,——————>
>—target_format—,—target_file—,—comment—)——————><

```

Syntax

Store content from server file with format conversion

```

>>> DB2Image—(—dbname—,—source_file—,—source_format—,——————>
>—target_format—,—target_file—,—comment—)——————><

```

Syntax

Store content from buffer or workstation client file with format conversion and additional changes

DB2Image

```
►►DB2Image—(—dbname—,—content—,—source_format—,——————→  
  
►—target_format—,—conversion_options—,—target_file—,—comment—)—————◄◄
```

Syntax

Store content from server file with format conversion and additional changes

```
►►DB2Image—(—dbname—,—source_file—,—source_format—,——————→  
  
►—target_format—,—conversion_options—,—target_file—,—comment—)—————◄◄
```

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 71 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 (VARCHAR(254))

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 (VARCHAR(254))

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 (VARCHAR(16384))

A comment to be stored with 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 72 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',

        '',
        '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 by 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'));;
```

DB2ImageA

Image	Audio	Video
X		

Stores the content of an image with user-supplied attributes in a database table. The image source can be in a client buffer, workstation client file, or server file. The image can be stored in the database table as a BLOB, or in a server file (referred to 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 with user-supplied attributes from buffer or workstation client file

```

>>>DB2ImageA(—dbname—,—content—,—target_file—,——————>
>—————>—————>—————>—————>—————>—————>—————>
>comment—,—attrs—,—format—,—thumbnail—)—————>>

```

Syntax

Store content with user-supplied attributes from server file

```

>>>DB2Image(—dbname—,—source_file—,—stortype—,—comment—,——————>
>—————>—————>—————>—————>—————>—————>—————>
>attrs—,—format—,—thumbnail—)—————>>

```

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.

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 71 for supported image formats.

target_file (VARCHAR(254))

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 (VARCHAR(254))

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 (VARCHAR(16384))

A comment to be stored with the image.

Return values (data type)

Handle of the image (DB2IMAGE)

Examples

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;
char hvImgattrs[100];
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs;
pimgattr->width=640;
pimgattr->height=480;

DBiPrepareAttrs(pimgattr);

DBEXEC SQL INSERT INTO EMPLOYEE VALUES(
    '128557',
    'Anita Jones',
    DB2IMAGEA(
        CURRENT SERVER,
        '/employee/images/ajones.bmp',
        :hvStorageType,
        'Anita''s picture',
        :hvImgattrs,
        'FormatI',
        '')
    );
```

DB2Video

Image	Audio	Video
		X

Stores the content of a video in a database table. The video source can be in a client buffer, workstation client file, or server file. The video can be stored in the database table as a BLOB, or in a server file (referred to 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 workstation client file

```
►►DB2Video—(—dbname—,—content—,—format—,—target_file—,—comment—)——►◄
```

Syntax

Store content from server file

```
►►DB2Video—(—dbname—,—source_file—,—format—,—stortype—,—comment—)——►◄
```

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. See Table 5 on page 71 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 (VARCHAR(254))

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 (VARCHAR(254))

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 (VARCHAR(16384))

A comment to be stored with 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 VARCHAR(254))),
      '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'));
```

DB2VideoA

Image	Audio	Video
		X

Stores the content of a video with user-supplied attributes in a database table. The video source can be in a client buffer, workstation 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 with user-supplied attributes from buffer or workstation client file

```

▶▶ DB2VideoA(—dbname—, —content—, —target_file—, —————→
               ▶—comment—, —attrs—, —format—, —compress_type—, —thumbnail—)————→◀◀

```

Syntax

Store content with user-supplied attributes from server file

```

▶▶ DB2VideoA(—dbname—, —source_file—, —stortype—, —comment—, —————→
               ▶—attrs—, —format—, —compress_type—, —thumbnail—)————→◀◀

```

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 will be stored in the same format as its source. See Table 5 on page 71 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 (VARCHAR(254))

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 (VARCHAR(254))

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 (VARCHAR(16384))

A comment to be stored with the video.

attrs (VARCHAR (4096) FOR BIT DATA)

The attributes of the video.

compress_type (VARCHAR (8))

The compression format of the video.

thumbnail (VARCHAR (16384) 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 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;
    char hvVidattrs[4000];
    char hvThumbnail[16384];
EXEC SQL END DECLARE SECTION;

MMDBVideoAttrs      *pvideoAttr;

hvStorageType=MMDB_STORAGE_TYPE_EXTERNAL;

pvideoAttr=(MMDBVideoAttrs *)hvVidattrs;

/* Generate thumbnail and assign to thumbnail variable */

EXEC SQL INSERT INTO EMPLOYEE VALUES(
    '128557',
    'Anita Jones',
    DB2VIDEOA(
        CURRENT SERVER,
        '/employee/videos/ajones.vid',
        :hvStorageType,
        'Anita''s video',
        :hvVidattrs,
        'FormatV',
        'MPEG1',
        :hvThumbnail)
    );
```

Duration

Duration

Image	Audio	Video
	X	X

Returns the duration (that is, the playing time in seconds) of a WAVE or AIFF audio, or video.

Include file

audio dmbaudio.h

video dmbvideo.h

Syntax

►►—Duration—(*—handle—*)—————►◄

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 the duration, in seconds, of 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 that are 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

►►—Filename—(—*handle*—)—————►◄

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(254)); 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[255];
EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL SELECT FILENAME(VIDEO)
INTO :hvVid_fname
FROM EMPLOYEE
WHERE NAME='Robert Smith';
```

FindInstrument

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

►►—FindInstrument—(—*handle*—,—*instrument*—)————►►

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

►►FindTrackName(—handle—,—trackname—)◄◄

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. The audio recording is 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

►►—Format—(—*handle*—)—————►►

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 71 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

►►—FrameRate—(*—handle—*)—————►◄

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 that is 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

GetInstruments

Image	Audio	Video
	X	

Returns instrument name of all instruments in a MIDI audio.

Include file

dmbaudio.h

Syntax

►► `GetInstruments`—(`—handle—`)—————►◄

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 instrument, 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. The audio recording is 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

►►—GetTrackNames—(*—handle—*)—————►

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

►► Height—(*—handle—*)—————►►

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[255];
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

►►—Importer—(—*handle*—)—————►◄

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[255];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
INTO :hvAud_fname
FROM EMPLOYEE
WHERE IMPORTER(SOUND)='rsmith';
```

ImportTime

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

►►—ImportTime—(—*handle*—)—————►

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[255];
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

►►—MaxBytesPerSec—(—*handle*—)————►◄

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 that is 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

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

►►—NumAudioTracks—(*—handle—*)—►►

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[255];
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

►►—NumChannels—(—*handle*—)—————►◄

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[255];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(SOUND)
  INTO :hvAud_fname
  FROM EMPLOYEE
  WHERE NUMCHANNELS(SOUND) = 2;
```

NumColors

NumColors

Image	Audio	Video
X		

Returns the number of colors in an image.

Include file

dmbimage.h

Syntax

►►—NumColors—(*--handle—*)—————►◄

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[255];
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

►►—NumFrames—(*—handle—*)—————►◄

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 that is 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

NumVideoTracks

Image	Audio	Video
		X

Returns the number of video tracks in a video.

Include file

dmbvideo.h

Syntax

►►—NumVideoTracks—(*—handle—*)—————►◄

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[255];
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.)

Notes:

1. **EEE Only:** QbScoreFromName is not supported in a partitioned database environment. Use the the QbScoreFromStr UDF instead, after using the QbQueryGetString API to get the query string.
2. QbScoreFromName will be deprecated in future releases for non-partitioned database environments. To reuse a query, you should use the QbQueryGetString API to get the query string and save that string for later use in your application.

Include file

none

Syntax

►► QbScoreFromName (—imgHandle—, —queryName—) ►►

Syntax

Deprecated version

►► QbScoreFromName (—queryName—, —imgHandle—) ►►

Parameters (data type)

imgHandle (DB2Image)

The handle of the image.

queryName (varchar(18))

The name of the query object.

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 a null value means that the image has not been cataloged; the deprecated version of this UDF returns score of -1 when the image has not been cataloged.

Examples

Find the cataloged images in a table column whose average color is very close to red:

QbScoreFromName

```
EXEC SQL BEGIN DECLARE SECTION;
char Img_fnd[100];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT NAME
      INTO :Img_fnd
      FROM FABRIC
      WHERE (QBSCOREFROMNAME(SWATCH_IMG,
                             'fshavgcol'))<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

►►QbScoreFromStr(—imgHandle—,—query—)◄◄

Syntax

Depricated version

►►QbScoreFromStr(—query—,—imgHandle—)◄◄

Parameters (data type)

imgHandle (DB2Image)

The handle of the image.

query (VARCHAR(1024))

The query string.

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 a null value means that the image has not been cataloged; the depricated version of this UDF returns score of -1 when 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(Swatch_Img,
    'QbColorFeatureClass color=<255, 0, 0>'))<0.1
```

QbScoreTBFromName

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.

Notes:

1. **EEE Only:** QbScoreTBFromName is not supported in a partitioned database environment. Use the the QbScoreFromStr UDF instead, after using the QbQueryGetString API to get the query string.
2. QbScoreTBFromName will be deprecated in the future for non-partitioned database environments. To reuse a query, you should use the QbQueryGetString API to get the query string and save that string for later use in your application.

Include file

none

Syntax

Return scores for all cataloged images in a column

►►QbScoreTBFromName(—queryName—,—table—,—column—)————►◄

Syntax

Return scores for a specific number of cataloged images in a column

►►QbScoreTBFromName(—queryName—,—table—,—column—,—maxReturns—)————►◄

Parameters (data type)

queryName (VARCHAR(18))

The name of the query object.

table (CHAR(18))

The qualified name of the table that contains the image column. You can use an unqualified table name if the table schema is the same as the user ID used to start DB2 extenders services.

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 that are 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 ascending 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 that is 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',
    'clothes.fabric',
    'swatch_img'))
AS T1));
```

QbScoreTBFromStr

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

►► QbScoreTBFromStr (—query—, —table—, —column—) ►►

Syntax

Return scores for a specific number of cataloged images in a column

►► QbScoreTBFromStr (—query—, —table—, —column—, —maxReturns—) ►►

Parameters (data type)

query (VARCHAR(1024))

The query string.

table (CHAR (18))

The qualified name of the table that contains the image column. You can use an unqualified table name if the table schema is the same as the user ID used to start DB2 extenders services.

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 ascending 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">'
           'clothes.fabric',
           'swatch_img',
           10))
        AS T1));

```

Replace

Replace

Image	Audio	Video
X	X	X

Updates the content of an image, audio, or video that is 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 workstation client file and update comment

```
►► Replace—(—handle—,—content—,—source_format—,——————►  
  
►—target_file—,—comment—)——————►◄
```

Syntax

Update content from server file and update comment

```
►► Replace—(—handle—,—source_file—,—source_format—,—stortype—,—————►  
  
►—comment—)——————►◄
```

Syntax

Update content from buffer or workstation client file with format conversion and update comment—image only

```
►► Replace—(—handle—,—content—,—source_format—,—————►  
  
►—target_format—,—target_file—,—comment—)——————►◄
```

Syntax

Update content from server file with format conversion and update comment—image only

```
►► Replace—(—handle—,—source_file—,—source_format—,—————►  
  
►—target_format—,—target_file—,—comment—)——————►◄
```


Syntax

Update content from buffer or workstation client file with format conversion and additional changes and update comment—image only

```
►►Replace—(—handle—,—content—,—source_format—,——————►
►target_format—,—target_file—,—conversion_options—,—comment—)—————►◄
```

Syntax

Update content from server file with format conversion and additional changes and update comment—image only

```
►►Replace—(—handle—,—source_file—,—source_format—,——————►
►target_format—,—conversion_options—,—target_file—,—comment—)—————►◄
```

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 (VARCHAR(254))

The name of the file that contains the content for the update of the image, audio, or video.

target_file (VARCHAR(254))

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 that 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.

Replace

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 (VARCHAR(16384))

A comment.

thumbnail (LONG VARCHAR FOR BIT DATA)

A thumbnail of the image or video frame (image and video only)

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';
```

ReplaceA

Image	Audio	Video
X	X	X

Updates the content of an image, audio, or video with user-supplied attributes that are stored in a database, and updates its comment.

Include file

image dmbimage.h
audio dmbaudio.h
video dmbvideo.h

Include file

Update content with user-supplied attributes from buffer or workstation client file and update comment

```
►► ReplaceA(—handle—,—content—,—target_file—,——————►
►comment—,—attrs—,—tracknames—,—instruments—,——————►
►format—,—compress_type—,—thumbnail—)——————◄◄
```

Syntax

Update content with user-supplied attributes from server file and update comment

```
►► ReplaceA(—handle—,—source_file—,—stortype—,—comment—,—————►
►attrs—,—tracknames—,—instruments—,—format—,—————►
►compress_type—,—thumbnail—)——————◄◄
```

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 (VARCHAR(254))

The name of the file that contains the content for the update of the image, audio, or video.

target_file (VARCHAR(254))

The name of the file that contains the content of the image, audio, or video to be updated.

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

ReplaceA

BLOB_LOCATOR. DB2 promotes the data type to BLOB_LOCATOR and passes the LOB locator to the Replace UDF.

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 (VARCHAR(16384))

A comment.

attrs (VARCHAR (4096) FOR BIT DATA)

The attributes of the image, audio, or video

tracknames (VARCHAR(1536))

The names of all tracks in the MIDI audio (audio only). The values are in track number order (for example, PIANO TUNE; TRUMPET FANFARE). If a track does not have a name, its field is blank. A null value should be specified for audio formats other than MIDI.

instruments (VARCHAR(1536))

The names of all instruments in the MIDI audio (audio only). The values are in track number order (for example, PIANO; TRUMPET; BASS). If a track does not have an associated instrument, its field is blank. A null value should be specified for audio formats other than MIDI.

compress_type (VARCHAR (8))

The compression format of the video (video only).

thumbnail (VARCHAR (16384) FOR BIT DATA)

A thumbnail of the image or video frame (image and video only)

Return values (data type)

The handle of the image, audio, or video to be updated (DB2IMAGE, DB2AUDIO, or DB2VIDEO).

Examples

Update the image stored for Anita Jones that is in the picture column of the employee table and update its comment. 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.

```
EXEC SQL BEGIN DECLARE SECTION;
long hvStorageType;
char hvImgattrs[100];
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS    *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;

pimgattr = (DB2IMAGEATTRS *) hvImgattrs;
pimgattr->width=640;
pimgattr->height=480;

DBiPrepareAttrs(pimgattr);

EXEC SQL UPDATE EMPLOYEE
  SET VIDEO=REPLACEA(
    PICTURE,
    '/employee/newimg/ajones.bmp',
    :hvStorageType,
```

```
        'A new image for Anita Jones',  
        :ImgAttrs,  
        'FormatI',  
        '')  
WHERE NAME='Anita Jones';
```

SamplingRate

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

►►—SamplingRate—(*—handle—*)—————►►

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[255];
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

►►—Size—(—handle—)—————►◄

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[255];  
EXEC SQL END DECLARE SECTION;  
  
EXEC SQL SELECT FILENAME(PICTURE)  
INTO :hvImg_fname  
FROM EMPLOYEE  
WHERE SIZE(PICTURE) > 310000;
```

Thumbnail

Thumbnail

Image	Audio	Video
X		X

Returns or updates a thumbnail-size version of an image or video frame that is stored in a database.

Include file

image dmbimage.h

video dmbvideo.h

Syntax

Retrieve a thumbnail

►►Thumbnail—(*—handle—*)—————►◄

Syntax

Update a thumbnail

►►Thumbnail—(*—handle—*,*—new_thumbnail—*)—————►◄

Parameters (data type)

handle (DB2IMAGE or DB2VIDEO)

Column name or host variable that contains the handle of the image or video.

new_thumbnail (VARCHAR (16384) FOR BIT DATA)

Source content for update of thumbnail

Return values (data type)

For retrieval, the content of the retrieved thumbnail (VARCHAR(16384) 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;
char hvThumbnail[16384];
EXEC SQL END DECLARE SECTION;

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

Update the thumbnail that is associated with Anita Jones's video in the employee table:

```
EXEC SQL BEGIN DECLARE SECTION;
char hvThumbnail[16384];
EXEC SQL END DECLARE SECTION;

/* Create thumbnail and */
/* store in hvThumbnail */
```



```
EXEC SQL UPDATE EMPLOYEE
  SET VIDEO=THUMBNAIL(
    VIDEO,
    :hvThumbnail)
  WHERE NAME='Anita Jones';
```

TicksPerQNotes

TicksPerQNote

Image	Audio	Video
	X	

Returns the clock speed of a recorded MIDI audio, in ticks per quarter note.

Include file

dmbaudio.h

Syntax

►►—TicksPerQNote—(*—handle—*)—————►◄

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[255];
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

►►—TicksPerSec—(*—handle—*)—————►

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[255];
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

►►—Updater—(—*handle*—)—————►►

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 that is 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

►►—UpdateTime—(—*handle*—)—————►

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[255];
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

►► **Width**—(*—handle—*)—————►►

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[255];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT FILENAME(PICTURE)
      INTO :hvImg_fname
      FROM EMPLOYEE
      WHERE WIDTH(PICTURE)<300;
```

Chapter 14. 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 provides 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 referred to in audio columns of user tables. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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, or SELECT privilege on enabled audio columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 server are searched.

count (out)

The number of entries in the output list.

fileList (out)

A list of inaccessible files that are referred to 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 server.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

MMDB_RC_NO_AUTH

User does not have access authority to the required tables.

MMDB_RC_WARN_NO_AUTH

User does not have access authority to some of the required tables.

Examples

List all inaccessible files that are referred to in audio columns of tables that are 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 referred to 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 that are allocated by this API after calling it. Specifically, you must free up the fileList data structure.

Authorization

SYSADM, or SELECT privilege on enabled audio columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 server are searched.

count (out)

The number of entries in the output list.

fileList (out)

A list of files that are referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

MMDB_RC_NO_AUTH

User does not have access authority to the required tables.

MMDB_RC_WARN_NO_AUTH

User does not have access authority to some of the required tables.

Examples

List all files that are referred to in audio columns in tables that are 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 refer to a specified file. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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, or SELECT privilege on enabled audio columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 server are searched.

fileName (in)

the name of the referred to file.

count (out)

The number of entries in the output list.

tableList (out)

A list of table entries that refer to the specified file.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

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 server that refer to file /audios/asmith.wav:

```
#include <dmbaudio.h>
long idx;

rc = DBaAdminIsFileReferenced(NULL,
    "/audios/asmith.wav",
    &count, &tableList);
```

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 server before calling this API. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 server.

Examples

Disable the sound column in the employee table for audio (DB2Audio data):

```
#include <dmbaudio.h>
```

```
rc = DBaDisableColumn("employee", "sound");
```

DBaDisableServer

Image	Audio	Video
	X	

Disables a database server for audio (DB2Audio data) so that it cannot hold audio data. All tables in the database server that is defined for DB2Audio are also disabled. The metadata and UDFs that are defined by the Audio Extender for the database server are dropped. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

Include file

dmbaudio.h

Syntax

```
long DBaDisableServer(
);
```

Parameters

DBaDisableServer 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 server.

Examples

Disable the database server for audio (DB2Audio data):

```
#include <dmbaudio.h>
```

```
rc = DBaDisableServer();
```


DBaDisableTable

Image	Audio	Video
	X	

Disables a table for audio (DB2Audio data) so that it cannot hold audio data. All columns in the table that is defined for DB2Audio are also disabled. Some of the metadata that is 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 server before calling this API. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 server.

DBaDisableTable

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 server. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Use privilege is also required on table spaces and buffer pools that are specified in the API parameters.

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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.

DBaEnableColumn

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 server.

MMDB_RC_NOT_ENABLED

Database server 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");
```

DBaEnableServer

Image	Audio	Video
	X	

Enables a database server for audio (DB2Audio data). This API is called once per database server. It defines a DB2 user-defined type, DB2Audio, to the database server. It also creates all UDFs that manipulate DB2Audio data. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

Include file

dmbaudio.h

Syntax

```
long DBaEnableServer(
    char *tableSpace,
    char *wlmNames,
    char *externalSecurity
);
```

Parameters

tableSpace (in)

The table space specification for administrative support tables. The specification has two parts as follows:

- The name of the table space for administrative support tables that store attribute data. The table space name must be the name of a table space that is defined in the MMDBSYS database. If you do not specify a table space name, DB2 creates a table space in the MMDBSYS database for each global administrative support table.
- For the table space, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults. For details about these blocks and index options, see the description of the CREATE INDEX command in the *SQL Reference*.

wlmNames (in)

WLM environment names. A maximum of two can be specified; at least one must be specified. If only one is specified, then all extender UDFs run in that WLM environment. If two WLM environments are specified, the second is used to run UDFs that store, retrieve, or update objects (such as

DBaEnableServer

DB2AUDIO, CONTENT, and REPLACE). The first WLM environment is used for attribute retrieval UDFs (such as FORMAT and DURATION).

externalSecurity

Indicates how the UDFs interact with an external security product, such as RACF, to control access to files. UDFs that use files include import and export UDFs such as DB2AUDIO and CONTENT, they do not include attribute retrieval UDFs such as FORMAT.

You can specify EXTERNAL SECURITY USER or EXTERNAL SECURITY DB2. If you specify EXTERNAL SECURITY USER, each UDF executes as if has the user ID (that is, the primary authorization ID) of the process that invoked it, and has permissions as defined for that user ID on the OS/390 server.

If you specify EXTERNAL SECURITY DB2, UDF access to files is performed using the authorization ID established for the WLM environment that runs file-accessing UDFs. All extender UDF invokers have access to the same files. EXTERNAL SECURITY DB2 is the default.

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 server is already enabled.

MMDB_RC_NOT_MMDBSYS_DBNAME

The table space does not exist in database MMDBSYS.

MMDB_RC_NO_TABLESPACE_SPECIFICATION

The table space is not specified.

MMDB_RC_CANNOT_SET_SQLID_TO_MMDBSYS

None of the authorization IDs of the application process has SYSADM authority and none of the authorization IDs of the application process is MMDBSYS.

Examples

Enable the database server for audio (DB2Audio data) in the table space MYTS. Specify WLM environment DMBWLM1. Use defaults for the index table space and external security specifications:

```
#include <dmbaudio.h>
```

```
rc = DBaEnableServer("myts","dmbwlm1",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 server. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS with a secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Use privilege is also required on table spaces and buffer pools that are specified in the API parameters.

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

Include file

dmbaudio.h

Syntax

```
long DBaEnableTable(
    char *tableSpace,
    char *tableName
);
```

Parameters

tableSpace (in)

The table space specification for administrative support tables and LOB data. The specification has four parts as follows:

- The name of the table space for administrative support tables that store attribute data. You must specify this table space. The table space name should be qualified by the database name; the table space should be in the same database as the user table.

DBaEnableTable

- For the table space for administrative support tables, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults.
- The name of the table space for LOB data. You must specify this table space. The table space name should be qualified by the database name; the table space should be in the same database as the user table.
- For the table space for LOB data, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults.

For details about the blocks and options for indexes, see the description of the CREATE INDEX command in the *SQL Reference*.

tableName (in)

The name of the table that will contain an audio column.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_WARN_ALREADY_ENABLED

Table is already enabled.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_TABLE_DOESNOT_EXIST

Table does not exist.

Examples

Enable the employee table for audio (DB2Audio data) in the table space MYTS. Specify table space MYLOBTS for LOB data and use defaults for the index table spaces:

```
#include <dmbaudio.h>

rc = DBaEnableTable("myts,,mylobts",
    "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/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 referred to in audio columns of user tables. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server before calling this API.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List all inaccessible files that are referred to 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 referred to 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 server before calling this API.

It is important that you free up the resources that are 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/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List all files that are referred to in audio columns in the employee table:

```
#include <dmbaudio.h>
long idx;

rc = DBaGetReferencedFiles("employee",
    &count, &filelist);
```

DBalsColumnEnabled

Image	Audio	Video
	X	

Determines whether a column has been enabled for audio (DB2Audio data). The application must be connected to a database server before calling this API.

Authorization

SYSADM, DBADM, table owner, or SELECT privilege on the user table

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 numeric 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 server.

Examples

Determine if the sound column in the employee table is enabled for audio:

```
#include <dmbaudio.h>
```

```
rc = DBaIsColumnEnabled("employee",  
    "sound", &status);
```

DBalsFileReferenced

Image	Audio	Video
	X	

Returns a list of table entries that refer to a specified file. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 referred to file.

count (out)

The number of entries in the output list.

tableList (out)

A list of table entries that refer to the specified file.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List the entries in audio columns of the employee table that refer to file
/audios/ajones.wav:

```
#include <dmbaudio.h>
long idx;

rc = DBaIsFileReferenced(NULL,
    "/audios/ajones.wav",
    &count, &tableList);
```

DBalsServerEnabled

Image	Audio	Video
	X	

Determines whether a database server has been enabled for audio (DB2Audio data). The application must be connected to a database server before calling this API.

Authorization

None

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

Include file

dmbaudio.h

Syntax

```
long DBaIsServerEnabled(
    short *status
);
```

Parameters

status (out)

Indicates whether the database server is enabled. This parameter returns a numeric 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 server.

Examples

Determine if the database server is enabled for audio:

```
#include <dmbaudio.h>

rc = DBaIsServerEnabled(&status);
```

DBaIsTableEnabled

Image	Audio	Video
	X	

Determines whether a table has been enabled for audio (DB2Audio data). The application must be connected to a database server before calling this API.

Authorization

None

Library file

OS/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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 numeric 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 server.

Examples

Determine if the employee table is enabled for audio (DB2Audio data):

DBaIsTableEnabled

```
#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.
- 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 server and have SELECT privilege on the administrative support tables for the user 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 that is 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/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

Include file

dmbaudio.h

Syntax

Play an audio stored in a column

```
long DBaPlay(
    char *playerName,
    MMDB_PLAY_HANDLE,
    DB2Audio *audioHandle,
    waitFlag
);
```

Syntax

Play an audio stored as a file

```
long DBaPlay(
    char *playerName,
    MMDB_PLAY_FILE,
    char *fileName,
    waitFlag
);
```

DBaPlay

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 server.

Examples

Play the audio that is 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 that is used by most UNIX and OS/390 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/390	AIX	Windows	Solaris
DMBAUDIO	libdmbaudio.a	dmbaudio.lib	libdmbaudio.so

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);
```

DBiAdminGetInaccessibleFiles

Image	Audio	Video
X		

Returns the names of inaccessible files that are referred to in image columns of user tables. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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, or SELECT privilege on enabled image columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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 referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_NO_AUTH

User does not have access authority to the required tables.

MMDB_RC_WARN_NO_AUTH

User does not have access authority to some of the required tables.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List all inaccessible files that are referred to in image columns of tables that are 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 referred to 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 server before calling this API.

It is important that you free up the resources that are allocated by this API after calling it. Specifically, you must free up the fileList data structure.

Authorization

SYSADM, or SELECT privilege on enabled image columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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 server are searched.

count (out)

The number of entries in the output list.

fileList (out)

A list of files that are referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_NO_AUTH

User does not have access authority to the required tables.

MMDB_RC_WARN_NO_AUTH

User does not have access authority to some of the required tables.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List all files that are referred to in image columns in tables that are 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 refer to a specified file. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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, or SELECT privilege on enabled image columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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 server are searched.

fileName (in)

The name of the referred to file.

count (out)

The number of entries in the output list.

tableList (out)

A list of table entries that refer to the specified file.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

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 refer to file /images/asmith.bmp:

```
#include <dmbimage.h>
long idx;

rc = DBiAdminIsFileReferenced(NULL,
    "/images/asmith.bmp",
    &count, &tableList);
```

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 server and have SELECT privilege on the administrative support tables for the user 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 that is 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/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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

```
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 server.

Examples

Display the image that is 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. The QBIC catalog that is associated with this column is also deleted. 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 server before calling this API. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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 server.

Examples

Disable the picture column in the employee table for images (DB2Image data):

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

```
rc = DBiDisableColumn("employee",  
    "picture");
```

DBiDisableServer

Image	Audio	Video
X		

Disables a database server for images (DB2Image data) so that it cannot hold image data. All tables in the database server that is defined for DB2Image are also disabled. The metadata and UDFs that are defined by the Image Extender for the database server are dropped. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

Include file

dmbimage.h

Syntax

```
long DBiDisableServer(
);
```

Parameters

DBiDisableServer 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 server.

Examples

Disable the database server for images (DB2Image data):

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

```
rc = DBiDisableServer();
```

DBiDisableTable

Image	Audio	Video
X		

Disables a table for images (DB2Image data) so that it cannot hold image data. All columns in the table that is defined for DB2Image are also disabled. Some of the metadata that is defined by the Image Extender for the table is dropped. All QBIC catalogs that are associated with the image columns in the table are also deleted. 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 server before calling this API. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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.

DBiDisableTable

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

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 server. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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.

DBiEnableColumn

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_WRONG_SIGNATURE

Datatype for the specified column is incorrect. User-defined type MMDBSYS.DB2IMAGE is expected.

MMDB_RC_COLUMN_DOESNOT_EXIST

Column is not defined in the specified table.

MMDB_RC_NOT_ENABLED

Database server or table is not enabled.

Examples

Enable the picture column in the employee table for images:

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

```
rc = DBiEnableColumn("employee",  
    "picture");
```

DBiEnableServer

Image	Audio	Video
X		

Enables a database server for images (DB2Image data). This API is called once per database server. It defines a DB2 user-defined type, DB2Image, to the database server. It also creates all UDFs that manipulate DB2Image data. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

Include file

dmbimage.h

Syntax

```
long DBiEnableServer(
    char *tableSpace,
    char *wlmNames,
    char *externalSecurity
);
```

Parameters

tableSpace (in)

The table space specification for administrative support tables. The specification has two parts as follows:

- The name of the table space for administrative support tables that store attribute data. The table space name must be the name of a table space that is defined in the MMDBSYS database. If you do not specify a table space name, DB2 creates a table space in the MMDBSYS database for each global administrative support table.
- For the table space, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults. For details about these blocks and index options, see the description of the CREATE INDEX command in the *SQL Reference*.

wlmNames (in)

WLM environment names. A maximum of two can be specified; at least one must be specified. If only one is specified, then all extender UDFs run in that WLM environment. If two WLM environments are specified, the second is used to run UDFs that import or export objects (such as

DBiEnableServer

DB2IMAGE, CONTENT, and REPLACE). The first WLM environment is used for attribute retrieval UDFs (such as WIDTH, HEIGHT, and SIZE).

externalSecurity

Indicates how the UDFs interact with an external security product, such as RACF, to control access to files. UDFs that use files include import and export UDFs such as DB2IMAGE and CONTENT, they do not include attribute retrieval UDFs such as FORMAT.

You can specify EXTERNAL SECURITY USER or EXTERNAL SECURITY DB2. If you specify EXTERNAL SECURITY USER, each UDF executes as if has the user ID (that is, the primary authorization ID) of the process that invoked it, and has permissions as defined for that user ID on the OS/390 server.

If you specify EXTERNAL SECURITY DB2, UDF access to files is performed using the authorization ID established for the WLM environment that runs file-accessing UDFs. All extender UDF invokers have access to the same files. EXTERNAL SECURITY DB2 is the default.

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 server is already enabled.

MMDB_RC_NOT_MMDBSYS_DBNAME

The table space does not exist in database MMDBSYS.

MMDB_RC_NO_TABLESPACE_SPECIFICATION

The table space is not specified.

MMDB_RC_CANNOT_SET_SQLID_TO_MMDBSYS

None of the authorization IDs of the application process has SYSADM authority, and none of the authorization IDs of the application process is MMDBSYS.

Examples

Enable the database server for images (DB2Image data) in the table space MYTS. Specify WLM environment DMBWLM1. Use defaults for the index table space and external security specifications:

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

```
rc = DBiEnableServer("myts","dmbwlm1",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 server. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Use privilege is also required on table spaces and buffer pools that are specified in the API parameters.

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

Include file

dmbimage.h

Syntax

```
long DBiEnableTable(
    char *tableSpace,
    char *tableName
);
```

Parameters

tableSpace (in)

The table space specification for administrative support tables and LOB data. The specification has four parts as follows:

- The name of the table space for administrative support tables that store attribute data. You must specify this table space. The table space name should be qualified by the database name; the table space should be in the same database as the user table. A 32 KB page buffer pool for the table space is required.

DBiEnableTable

- For the table space for administrative support tables, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults.
- The name of the table space for LOB data. You must specify this table space. The table space name should be qualified by the database name; the table space should be in the same database as the user table.
- For the table space for LOB data, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults.

For details about the blocks and options for indexes, see the description of the CREATE INDEX command in the *SQL Reference*.

tableName (in)

The name of the table that will contain an image column.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_WARN_ALREADY_ENABLED

Table is already enabled.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_TABLE_DOESNOT_EXIST

Table does not exist.

Examples

Enable the employee table for images (DB2Image data) in the table space MYTS. Specify table space MYLOBTS for LOB data and use defaults for the index table spaces:

```
#include <dmbimage.h>

rc = DBiEnableTable("myts,,mylobts",
    "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/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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 referred to in image columns of user tables. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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 referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List all inaccessible files that are referred to in image columns in the employee table:

```
#include <dmbimage.h>
long idx;

rc = DBiGetInaccessibleFiles("employee",
                             &count, &filelist);
```

DBiGetReferencedFiles

Image	Audio	Video
X		

Returns the names of files that are referred to 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 server before calling this API.

It is important that you free up the resources that are 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/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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 referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List all files that are referred to in image columns in the employee table:

```
#include <dmbimage.h>
long idx;

rc = DBiGetReferencedFiles("employee",
    &count, &filelist);
```

DBIsColumnEnabled

Image	Audio	Video
X		

Determines whether a column has been enabled for images (DB2Image data). The application must be connected to a database server before calling this API.

Authorization

SYSADM, DBADM, table owner, or SELECT privilege on the user table

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

Include file

dmbimage.h

Syntax

```
long DBIsColumnEnabled(
    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 numeric 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_WARN_ALREADY_ENABLED

Column is already enabled.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

Examples

Determine if the picture column in the employee table is enabled for images:

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

```
rc = DBIsColumnEnabled("employee",  
    "picture", &status);
```

DBIsFileReferenced

Image	Audio	Video
X		

Returns a list of table entries in image columns that refer to a specified file. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

Include file

dmbimage.h

Syntax

```
long DBIsFileReferenced(
    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 referred to file.

count (out)

The number of entries in the output list

tableList (out)

A list of table entries that refer to the specified file

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List the entries in image columns of the employee table that refer to file
/images/ajones.bmp:

```
#include <dmbimage.h>
long idx;

rc = DBIsFileReferenced(NULL,
    "/images/ajones.bmp",
    &count, &tableList);
```

DBIsServerEnabled

Image	Audio	Video
X		

Determines whether a database server has been enabled for images (DB2Image data). The application must be connected to a database server before calling this API.

Authorization

None

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

Include file

dmbimage.h

Syntax

```
long DBIsServerEnabled(
    short *status
);
```

Parameters

status (out)

Indicates whether the database server is enabled. This parameter returns a numeric 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 server.

Examples

Determine if the database server is enabled for images:

```
#include <dmbimage.h>

rc = DBIsServerEnabled(&status);
```

DBIsTableEnabled

Image	Audio	Video
X		

Determines whether a table has been enabled for images (DB2Image data). The application must be connected to a database server before calling this API.

Authorization

None

Library file

OS/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

Include file

dmbimage.h

Syntax

```
long DBIsTableEnabled(
    char *tableName,
    short *status
);
```

Parameters

tableName (in)

A table name.

status (out)

Indicates whether the table is enabled. This parameter returns a numeric 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 server.

Examples

Determine if the employee table is enabled for images:

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

```
rc = DBIsTableEnabled("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 that is used by most UNIX and OS/390 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/390	AIX	Windows	Solaris
DMBIMAGE	libdmbimage.a	dmbimage.lib	libdmbimage.so

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);
```

DBvAdminGetInaccessibleFiles

Image	Audio	Video
		X

Returns the names of inaccessible files that are referred to in video columns of user tables. The application must be connected to a database server before calling this API

It is important that you free up the resources that are 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, or SELECT privilege on enabled video columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server are searched.

count (out)

The number of entries in the output list.

fileList (out)

A list of inaccessible files that are referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_NO_AUTH

User does not have access authority to the required tables.

DBvAdminGetInaccessibleFiles

MMDB_RC_WARN_NO_AUTH

User does not have access authority to some of the required tables.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List all inaccessible files that are referred to in video columns of tables that are 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 referred to 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 server before calling this API.

It is important that you free up the resources that are allocated by this API after calling it. Specifically, you must free up the fileList data structure.

Authorization

SYSADM, or SELECT privilege on enabled video columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server are searched.

count (out)

The number of entries in the output list.

fileList (out)

A list of files that are referred to in the table.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

DBvAdminGetReferencedFiles

MMDB_RC_NO_AUTH

User does not have access authority to the required tables.

MMDB_RC_WARN_NO_AUTH

User does not have access authority to some of the required tables.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

Examples

List all files that are referred to in video columns in tables that are 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 refer to a specified file. The application must be connected to a database server before calling this API.

It is important that you free up the resources that are 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, or SELECT privilege on enabled video columns in all searched user tables and associated administrative support tables

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server are searched.

fileName (in)

The name of the referred to file.

count (out)

The number of entries in the output list.

tableList (out)

A list of table entries that refer to the specified file.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

DBvAdminIsFileReferenced

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 refer to file /videos/asmith.mpg:

```
#include <dmbvideo.h>
long idx;

rc = DBvAdminIsFileReferenced(NULL,
    "/videos/asmith.mpg",
    &count, &tableList);
```

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 server before calling this API. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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.

DBvDisableColumn

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 server.

Examples

Disable the tv_ads column in the employee table for video (DB2Video data):

```
#include <dmbvideo.h>
```

```
rc = DBvDisableColumn("employee",  
    "tv_ads");
```

DBvDisableServer

Image	Audio	Video
		X

Disables a database server for video (DB2Video data) so that it cannot hold video data. All tables in the database server defined for DB2Video are also disabled. The metadata and UDFs defined by the Video Extender for the database server are dropped. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

Include file

dmbvideo.h

Syntax

```
long DBvDisableServer(
);
```

Parameters

DBvDisableServer 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 server.

Examples

Disable the database server for video (DB2Video data):

```
#include <dmbvideo.h>
```

```
rc = DBvDisableServer();
```

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 server before calling this API. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server.

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 server. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Use privilege is also required on table spaces and buffer pools specified in the API parameters.

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server.

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 server or table is not enabled.

Examples

Enable the video column in the employee table for video:

```
#include <dmbvideo.h>
```

```
rc = DBvEnableColumn("employee",  
    "video");
```

DBvEnableServer

Image	Audio	Video
		X

Enables a database server for video (DB2Video data). This API is called once per database server. It defines a DB2 user-defined type, DB2Video, to the database server. It also creates all UDFs that manipulate DB2Video data. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

Include file

dmbvideo.h

Syntax

```
long DBvEnableServer(
    char *tableSpace,
    char *wlmNames,
    char *externalSecurity
);
```

Parameters

tableSpace (in)

The table space specification for administrative support tables. The specification has two parts as follows:

- The name of the table space for administrative support tables that store attribute data. The table space name must be the name of a table space that is defined in the MMDBSYS database. If you do not specify a table space name, DB2 creates a table space in the MMDBSYS database for each global administrative support table.
- For the table space, the using-block, and/or free block, and/or gbpccache-block, and/or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults. For details about these blocks and index options, see the description of the CREATE INDEX command in the *SQL Reference*.

wlmNames (in)

WLM environment names. A maximum of two can be specified; at least one must be specified. If only one is specified, then all extender UDFs execute in that WLM environment. If two WLM environments are specified, the second is used to execute UDFs that import or export objects

(such as DB2VIDEO, CONTENT, and REPLACE); the first WLM environment is used for attribute retrieval UDFs (such as FORMAT, and DURATION).

externalSecurity

Indicates how the UDFs interact with an external security product, such as RACF, to control access to files. UDFs that use files include import and export UDFs such as DB2VIDEO and CONTENT, they do not include attribute retrieval UDFs such as FORMAT.

You can specify EXTERNAL SECURITY USER or EXTERNAL SECURITY DB2. If you specify EXTERNAL SECURITY USER, each UDF executes as if has the user ID (that is, the primary authorization ID) of the process that invoked it, and has permissions as defined for that user ID on the OS/390 server.

If you specify EXTERNAL SECURITY DB2, UDF access to files is performed using the authorization ID established for the WLM environment that executes file-accessing UDFs. All extender UDF invokers have access to the same files. EXTERNAL SECURITY DB2 is the default.

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 server is already enabled.

MMDB_RC_NOT_MMDBSYS_DBNAME

The table space does not exist in database MMDBSYS.

MMDB_RC_NO_TABLESPACE_SPECIFICATION

The table space is not specified.

MMDB_RC_CANNOT_SET_SQLID_TO_MMDBSYS

None of the authorization IDs of the application process has SYSADM authority and none of the authorization IDs of the application process is MMDBSYS.

Examples

Enable the database server for video (DB2Video data) in the table space MYTS. Specify WLM environment DMBWLM1. Use defaults for the index table space and external security specifications:

```
#include <dmbvideo.h>
```

```
rc = DBvEnableServer("myts","dmbwlm1",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 server. It is recommended that after calling this API you issue an SQL COMMIT statement. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Use privilege is also required on table spaces and buffer pools specified in the API parameters.

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

Include file

dmbvideo.h

Syntax

```
long DBvEnableTable(
    char *tableSpace,
    char *tableName
);
```

Parameters

tableSpace (in)

The table space specification for administrative support tables and LOB data. The specification has four parts as follows:

- The name of the table space for administrative support tables that store attribute data. You must specify this table space. The table space name should be qualified by the database name; the table space should be in the same database as the user table. A 32 KB page buffer pool for the table space is required

- For the table space for administrative support tables, the using-block and/or free block, and/or gbpcache-block, and/or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults.
- The name of the table space for LOB data. You must specify this table space. The table space name should be qualified by the database name; the table space should be in the same database as the user table.
- For the table space for LOB data, the using-block and/or free block, and/or gbpcache-block, and/or index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults.

For details about the blocks and options for indexes, see the description of the CREATE INDEX command in the *SQL Reference*.

tableName (in)

The name of the table that will contain a video column.

Error codes

MMDB_SUCCESS

API call processed successfully.

MMDB_WARN_ALREADY_ENABLED

Table is already enabled.

MMDB_RC_NOT_CONNECTED

Application does not have valid connection to a database server.

MMDB_RC_TABLE_DOESNOT_EXIST

Table does not exist.

Examples

Enable the employee table for video (DB2Video data) in the table space MYTS. Specify table space MYLOBTS for LOB data and use defaults for the index table spaces:

```
#include <dmbvideo.h>

rc = DBvEnableTable("myts,,mylobts",
    "employee");
```

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/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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);
```


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 server 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/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server.

MMDB_RC_MALLOC

System cannot allocate memory to return the results.

DBvGetInaccessibleFiles

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 server 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/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server.

DBvGetReferencedFiles

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);
```

DBvIsColumnEnabled

Image	Audio	Video
		X

Determines whether a column has been enabled for video (DB2Video data). The application must be connected to a database server before calling this API.

Authorization

SYSADM, DBADM, table owner, or SELECT privilege on the user table

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server.

DBvIsColumnEnabled

Examples

Determine if the video column in the employee table is enabled for video:

```
#include <dmbvideo.h>
```

```
rc = DBvIsColumnEnabled("employee",  
                        "video", &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 server 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/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server.

DBvIsFileReferenced

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);
```


DBvIsServerEnabled

Image	Audio	Video
		X

Determines whether a database server has been enabled for video (DB2Video data). The application must be connected to a database server before calling this API.

Authorization

None

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

Include file

dmbvideo.h

Syntax

```
long DBvIsServerEnabled(
    short *status
);
```

Parameters

status (out)

Indicates whether the database server 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 server.

Examples

Determine if the database server is enabled for video:

```
#include <dmbvideo.h>

rc = DBvIsServerEnabled(&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 server before calling this API.

Authorization

None

Library file

OS/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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 server.

Examples

Determine if the employee table is enabled for video:

```
#include <dmbvideo.h>
```

```
rc = DBvIsTableEnabled("employee",
    &status);
```

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 server and have SELECT privilege on the administrative support tables for the user 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/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

Include file

dmbvideo.h

Syntax

Play a video stored in a column

```
long DBvPlay(
    char *playerName,
    MMDB_PLAY_HANDLE,
    DB2Video *videoHandle,
    waitFlag
);
```

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 server.

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 and OS/390 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/390	AIX	Windows	Solaris
DMBVIDEO	libdmbvideo.a	dmbvideo.lib	libdmbvideo.so

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);
```

QbAddFeature

Image	Audio	Video
X		

Adds a feature to the currently opened catalog. QbAddFeature creates the feature table for the specified feature in the database server. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work. After adding images to the image column in your user table, use the QbReCatalogColumn API, which adds an entry for each image to the feature table and analyzes the images.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER and SELECT privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Use privilege is also required on table spaces and buffer pools specified when the catalog is created.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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

Error codes

qbicECInvalidHandle

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. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM on the database
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has SELECT privilege on the user table
- Table owner
- Select privilege on the user table and SELECT, INSERT, UPDATE, and DELETE privileges on the administrative support tables for the enabled table and the QBIC catalog

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN QbCatalogColumn(
    QbCatalogHandle cHdl
);
```

Parameters

cHdl (in)

A pointer to the handle of the catalog.

Error codes

qbicECInvalidHandle

The catalog handle is not valid.

qbicECInvalidCatalog

The specified handle or table column is not valid for the catalog.

qbicECCatalog Errors

Errors occurred while cataloging individual images, these error were logged. Rollback not incurred.

qbicECImageNotFound

The image cannot be found or accessed.

qbicECCatalogRO

The catalog is read-only.

qbicECSQLError

An SQL error occurred.

Examples

```
#include <dmbqbapi.h>

rc = QbCatalogColumn(CatHd1);
```

QbCloseCatalog

Image	Audio	Video
X		

Closes the catalog. The API frees the opened catalog handle and the allocated resources.

Authorization

None

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN QbCloseCatalog(
    QbCatalogHandle cHdl
);
```

Parameters

cHdl (in)
A pointer to the handle of the catalog.

Error codes

qbicECInvalidHandle
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 server 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. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER and SELECT privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Use privilege is also required on table spaces and buffer pools specified in the API's parameters.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN QbCreateCatalog(
    char *tableName,
    char *columnName,
    SQLINTEGER autoCatalog,
    char *tableSpace
);
```

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)

This value must be set to 0 to indicate manual cataloging. Manual cataloging means that the user will explicitly request the Image Extender to catalog images. You can use the QbCatalogColumn API to catalog images that you add to the image column. (By comparison, the Image Extender for UDB Version 5.2, that is, the workstation version of the Image Extender,

QbCreateCatalog

supports automatic cataloging. Automatic cataloging means that the Image Extender automatically catalogs an image after the image is stored in a user table.

tableSpace (in)

The table space and index options for the QBIC catalog. The specification has four parts:

- The name of the table space for the catalog tables that contain feature data. The table space must be specified. The table space should be a segmented table space.
- For the index created on the catalog tables, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. This specification is optional. You get defaults if you do not specify this part.
- The name of the table space for the catalog log table. The table space can be a simple table space or a segmented table space. This specification is optional. If you do not specify a table space for the log table, the table space specified for the feature data tables is used.
- For the index created on the log data table, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. This specification is optional. You get defaults if you do not specify this part.

Error codes

qbicECSqlError

An SQL error occurred.

qbicECNotEnabled

The database server, table, or column is not enabled for the DB2Image data type.

qbicECDupCatalog

The catalog already exists.

qbicECUnsupportedOption

An unsupported option was specified.

qbicECErrorParameterTooLong

A parameter was too long for processing.

qbicECqerr

A QBIC error occurred, a message was produced.

qbicECqerrUnknown

An internal QBIC error occurred, a generic error message was produced.

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 server. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER and SELECT privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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

- qbicECInvalidHandle**
The catalog handle is not valid.
- qbicECCatalogInUse**
The catalog was being used by someone else.
- qbicECCatalogRO**
The catalog is read-only.
- qbicECSysmtem**
A system error occurred.

QbDeleteCatalog

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.
- The manual cataloging indicator.
- The table specifications for the catalog.

Authorization

- SYSADM
- DBADM on the database
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has SELECT privilege on the user table
- SELECT privilege on the user table and SELECT privilege on the QBIC catalog tables

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN QbGetCatalogInfo(
    QbCatalogHandle cHdl,
    QbCatalogInfo *catInfo
);
```

Parameters

cHdl (in)

A pointer to the handle of the catalog.

catInfo (out)

The catalog information structure.

Error codes

qbicECInvalidHandle

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

- SYSADM
- DBADM on the database
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS
- SELECT privilege on the QBIC Catalog tables

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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

qbicECInvalidHandle

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

None

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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 QBIC Catalog.

Authorization

None.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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 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.

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/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN QbQueryCreate(
    QbQueryHandle *qObj
);
```

Parameters

qObj (out)

A pointer to the query handle. If unsuccessful, this handle is set to 0.

Error codes

qbiEAllocation

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 an unnamed query object. The API releases all the memory used by the query object and any added features.

Authorization

None.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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 following features are supplied with the Image Extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Authorization

None.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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 object identified by the handle qoHandle:

```
#include <dmbqbapi.h>
```

```
rc = QbQueryGetFeatureCount(qoHandle,
    &count);
```


QbQueryGetString

Image	Audio	Video
X		

Returns the query string from a query. You can use the query string for input to UDFs in your application, for example in the UDF QbScoreFromStr or the API QbQueryStringSearch.

Authorization

None.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN QbQueryGetString(
    QbQueryHandle qObj,
    (char*)* queryString
);
```

Parameters

qObj (in)

The handle of the query object.

queryString (out)

The pointer to the query string for the query object.

Error codes

qbiECinvalidQueryHandle

The query handle that you specified does not reference a valid query.

Examples

Return the query string for the query object identified by the handle qrHandle.

```
#include <dmbqbapi.h>
```

```
SQLRETURN rc;
char *queryString;
QbQueryHandle qrHandle
```

```
rc = QbQueryGetString(qrHandle, &queryString);
if (rc == 0) {
    ... /* use the returned queryString for input to UDFs */
    free((void*)queryString); /* you must free queryString */
    queryString=(char*)0;
}
```

QbQueryListFeatures

Image	Audio	Video
X		

Returns a current list of features in the query object. The API returns the list to a buffer that you allocate. The following features are supplied with the Image Extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Authorization

None.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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 provide the name and can provide the description of the query object.

Notes:

1. **EEE Only:** QbQueryNameCreate is not supported in a partitioned database environment.
2. QbQueryNameCreate will be deprecated in future releases for non-partitioned database environments. To save a query, you should use QbQueryGetString to get the query string and save that string for later use in your application.

Authorization

Insert authority on the MMDBSYS.QBICQUERIES table.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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.

Notes:

1. **EEE Only:** QbQueryNameDelete is not supported in a partitioned database environment.
2. QbQueryNameDelete will be deprecated in future releases for non-partitioned database environments.

Authorization

Delete authority on the MMDBSYS.QBICQUERIES table.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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.

Notes:

1. **EEE Only:** QbQueryNameSearch is not supported in a partitioned database environment.
2. QbQueryNameSearch will be deprecated in future releases for non-partitioned database environments. To save a query, you should use QbQueryGetString to get the query string and save that string for later use in your application.

Authorization

- SELECT privilege on MMDBSYS.QBICQUERIES
- SELECT privilege on the QBIC catalog tables
- SELECT privilege on administrative support tables for any handles specified in the query
- Authority to reference files specified in the query

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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.

QbQueryNameSearch

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/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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:

QbQueryRemoveFeature

```
#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 privilege on the QBIC catalog tables
- SELECT privilege on administrative support tables for any handles specified in the query
- Authority to reference files specified in the query

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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.

QbQuerySearch

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 that you allocate the buffer large enough to hold all the results that 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 user table, . In addition, you can explicitly specify data for the average color or histogram color feature.

Use the QbQueryStringSearch following setting the source for image data in a server file using QbQuerySetFeatureData. QbQuerySearch does not use the source for image data from a server file set with QbQuerySetFeatureData.

The following features are supplied with the Image extender:

- QbColorFeatureClass
- QbColorHistogramFeatureClass
- QbDrawFeatureClass
- QbTextureFeatureClass

Authorization

None.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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 127 for more information.

QbQuerySetFeatureData

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 the histogram color feature in a query object. The data source for the feature is an image in a users table:

```
#include <dmbqbapi.h>
```

```
QbQueryHandle qoHandle;  
QbImageSource imgSource;
```

```
imgSource.sourceType = qbiSource_ImageHandle;  
strcpy(imgSource.imageHandle,handleFile);
```

```
rc = QbQuerySetFeatureData(qoHandle, "QbColorHistogramFeatureClass", &imgSource);
```

QbQuerySetFeatureWeight

Image	Audio	Video
X		

Sets the weight of the specified feature in a query object.

Authorization

None.

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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 average 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 privilege on the QBIC catalog tables
- SELECT privilege on administrative support tables for any handles specified in the query
- Authority to reference files specified in the query

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN QbQueryStringSearch(
    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		

Reanalyze 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. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or a user with secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER and SELECT privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN QbReCatalogColumn (
    QbCatalogHandle cHdl
);
```

Parameters

cHdl (in)

A pointer to the handle of the catalog.

Error codes

qbicECInvalidHandle

The catalog handle is not valid.

qbicECInvalidCatalog

The specified handle or table column is not valid for the catalog.

qbicECCatalog Errors

Errors occurred while cataloging individual images, these error were logged. Rollback not incurred.

qbicECImageNotFound

The image cannot be found or accessed.

qbicECCatalogRO

The catalog is read-only.

qbicECSQLError

An SQL error occurred.

Examples

Reanalyze all existing images in the opened QBIC catalog for a new feature:

```
#include <dmbqbapi.h>
```

```
rc = QbReCatalogColumn(CatHdl);
```

QbRemoveFeature

Image	Audio	Video
X		

Deletes the specified feature from the opened catalog. This API can roll back the unit of work in response to a severe error, or if the steps in a multiple step job are not completed. As a result, uncommitted work will be lost. In general, this API should exist in its own unit of work.

Authorization

Alter

Library file

OS/390	AIX	Windows	Solaris
DMBQBAPI	libdmbqbapi.a	dmbqbapi.lib	libdmbqbapi.so

Include file

dmbqbapi.h

Syntax

```
SQLRETURN 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

qbicECInvalidHandle

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");
```

QbRemoveFeature

Chapter 15. 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 for workstation clients only, 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 workstation DB2 command prompt or the MVS TSO DSN command processor.

To start the db2ext command-line processor in interactive mode, do the following:

Client	Action
AIX, Solaris	Enter the DB2EXT command from the operating system command prompt.
Windows 95, Windows 98, Windows NT, Window 2000	Double-click on the DB2EXT Command Line Processor icon in the DB2 Extenders folder, or enter the DB2EXT command from the DB2 command window.
OpenEdition	Issue the command: db2ext -a subsystem_ID where subsystem_ID is the identification of the pertinent DB2 subsystem (for example, V61A).
TSO	Issue the TSO call: call 'loadlib' '-a subsystem_ID' asis where loadlib is the load-member specification for the db2ext-command line processor, for example, dmb.v61a.Loadlib(db2ext), and subsystem_ID is the identification of the pertinent DB2 subsystem (for example, V61A).

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 (workstation client only), enter them from the operating system command line. You must precede each DB2 extender command with db2ext, for example:
db2ext enable server for db2image

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

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER and SELECT privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Command syntax

►►—ADD QBIC FEATURE—*feature_name*—————►►

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

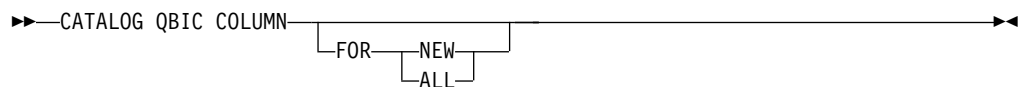
Image	Audio	Video
X		

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

- SYSADM
- DBADM on the database
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has SELECT privilege on the user table
- Table owner
- Select privilege on the user table and SELECT, INSERT, UPDATE, and DELETE privileges on the administrative support tables for the enabled table and the QBIC catalog

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 for new
```

Usage notes

When NEW is specified, the Image Extender updates the catalog only with the images that have not been cataloged. When ALL is specified, the Image 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

►►—CLOSE QBIC CATALOG—◀◀

Command parameters

None.

Examples

Close the current catalog:

```
close qbic catalog
```

Usage notes

The QBIC catalog must be open.

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

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER and SELECT privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Command syntax

```
►►—CREATE QBIC CATALOG—table_name—column_name—OFF—USING—tablespace_name—►◄
```

Command parameters

table_name

The name of the DB2IMAGE enabled table.

column_name

The name of the DB2IMAGE enabled column.

OFF Images are manually cataloged.

tablespace_name

The table space specification and index options for the QBIC Catalog. The specification has four parts:

- The name of the table space for the catalog tables that contain feature data. The table space must be specified. The table space should be a segmented table space.
- For the index created on the catalog tables, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. This specification is optional. You get defaults if you do not specify this part.
- The name of the table space for the catalog log table. The table space can be a simple table space or a segmented table space. This specification is optional. If you do not specify a table space for the log table, the table space specified for the feature data tables is used.
- For the index created on the log data table, any combination of the using-block, free block, gbpcache-block, or index options for type 2 non-partitioned indexes. This specification is optional. You get defaults if you do not specify this part.

Examples

Create a QBIC catalog for the picture column in the employee table. Use table space qbtbspace for the catalog:

```
create qbic catalog employee picture off qbtbspace
```

Usage notes

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.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER and SELECT privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Command syntax

►►—DELETE QBIC CATALOG—*table_name*—*column_name*—►►

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 with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Command syntax

```
➤—DISABLE COLUMN—table_name—col_name—FOR—extender_name—➤
```

Command parameters

table_name

The name of the table in the current database server.

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 SERVER

DISABLE SERVER


Image	Audio	Video
X	X	X

Disables the current database server from storing media data.

Authorization

- SYSADM
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS

Command syntax

►►—DISABLE SERVER FOR—extender_name◄◄

Command parameters

extender_name

The name of the extender for which you want to disable the current database server. Valid extender names are db2image, db2audio, and db2video.

Examples

Disable the current database server from holding image data:

```
disable server for db2image
```

Usage notes

Connect to the database server before using this command.

When you disable a database server, the system:

- Disables all the tables that are enabled for the specified extender.
- Drops the UDFs and 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 with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS or secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Command syntax

```

▶▶—DISABLE TABLE—table_name—FOR—extender_name—▶▶

```

Command parameters

table_name

The name of the table you want to disable in the current database server.

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 server 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.

ENABLE COLUMN

ENABLE COLUMN

Image	Audio	Video
X	X	X

Enables the specified column to store media data.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Command syntax

►►—ENABLE COLUMN—*table_name*—*col_name*—FOR—*extender_name*—————►◄

Command parameters

table_name

The name of the table in the current database server.

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 server before using this command.

ENABLE SERVER

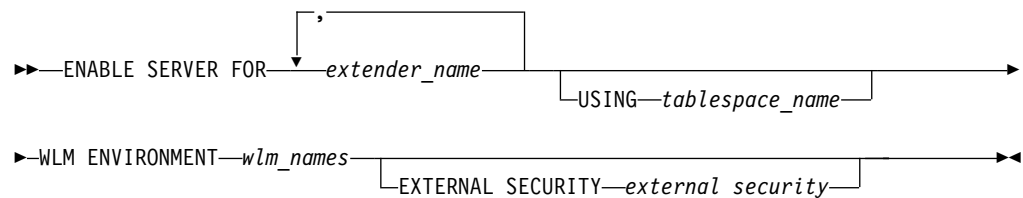
Image	Audio	Video
X	X	X

Enables the current database server to store media data using the specified table space.

Authorization

- SYSADM
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS

Command syntax



Command parameters

extender_name

The name of the extender for which you want to enable the current database server. Valid extender names are db2image, db2audio, and db2video.

tablespace_name

The name of the table spaces in which administrative support tables and their indexes are stored. The table space specification has two parts as follows:

- The name of the table space. The name must be the name of a table space that is defined in the MMDBSYS database. (The MMDBSYS database is created as part of the setup that is done after the DB2 extenders are installed.) For further information about DB2 extender installation and setup procedures, see the *Program Directory*.) If you do not specify a table space name, DB2 creates a table space in the MMDBSYS database for each global administrative support table.
- Any combination of using-block, free block, gbpcache-block, and index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults. For details about these blocks and index options, see the description of the CREATE INDEX command in the *SQL Reference*.

The parts of the table space specification are separated by a comma.

wlm_names

WLM environment names. A maximum of two can be specified (separated by a comma); at least one WLM environment name must be specified. If only one is specified, then all extender UDFs execute in that WLM environment. If two WLM environments are specified, the second is used to execute UDFs that import or export objects (such as DB2AUDIO,

ENABLE SERVER

CONTENT, and REPLACE); the first WLM environment is used for attribute retrieval UDFs (such as FORMAT and DURATION).

external_security

Indicates how the UDFs interact with an external security product, such as RACF, to control access to files. UDFs that use files include import and export UDFs such as DB2AUDIO and CONTENT, they do not include attribute retrieval UDFs such as FORMAT.

You can specify USER or DB2. If you specify USER, each UDF executes as if it has the user ID (that is, the primary authorization ID) of the process that invoked it, and has permissions as defined for that user ID on the OS/390 server.

If you specify DB2, UDF access to files is performed using the authorization ID established for the WLM environment that executes file-accessing UDFs. All extender UDF invokers have access to the same files. DB2 is the default.

Examples

Enable the current database server to hold image data in table space MMTBSYSG. Specify WLM environment WLMENV1 and EXTERNAL SECURITY DB2. Use the default for the type 2 non-partitioned index spaces:

```
enable server for db2image using mmtbsysg wlm environment wlmenv1
external security db2
```

Usage notes

Connect to the database server before using this command.

ENABLE TABLE

Image	Audio	Video
X	X	X

Enables the specified table to store media data using the specified table space.

Authorization

- SYSADM
- DBADM with GRANT privilege, and CREATEIN and DROPIN privilege on the schema MMDBSYS
- User ID of MMDBSYS with secondary authorization ID of MMDBSYS; MMDBSYS ID has TRIGGER, SELECT, UPDATE, and DELETE privileges on the user table; SQL authorization ID for the process has CREATAB privilege on the target database or is the DBADM for the database

Use privilege is also required on table spaces and buffer pools specified in the command parameters.

Command syntax

```
►►—ENABLE TABLE—table_name—FOR—extender_name—USING—tablespace_name—►►
```

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 spaces into which administrative support tables and LOB data are stored. The table space specification has four parts as follows:

- The name of the table space for the administrative support tables. You must specify this table space. A 32 KB page buffer pool for the table space is required
- For the table space for administrative support tables, any combination of the using-block, free block, gbpcache-block, and index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults.
- The name of the table space for LOB data. You must specify this table space.
- For the table space for LOB data, any combination of the using-block, free block, gbpcache-block, and index options for type 2 non-partitioned indexes. You can specify a NULL value for defaults.

The parts of the table space specification are separated by commas.

Examples

Enable the table employee to hold image data in table space MYTS. Use the table space MYLOBTS for LOB data. Use defaults for block and index specifications:

ENABLE TABLE

```
enable table employee for db2image using myts,,mylobts
```

Usage notes

Connect to the database server before using this command.

GET EXTENDER STATUS

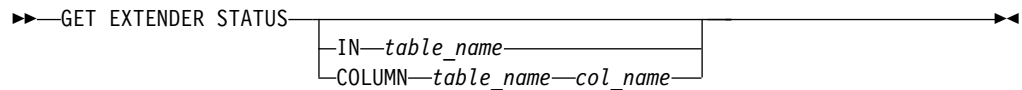
Image	Audio	Video
X	X	X

Displays the names of the extenders, if any, for which a column, table, or the current database server is enabled.

Authorization

None

Command syntax



Command parameters

table_name

The name of the table in the current database.server.

col_name

The name of the column.

Examples

Display the names of the enabled extenders in the database server:

```
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 server before using this command.

GET INACCESSIBLE FILES

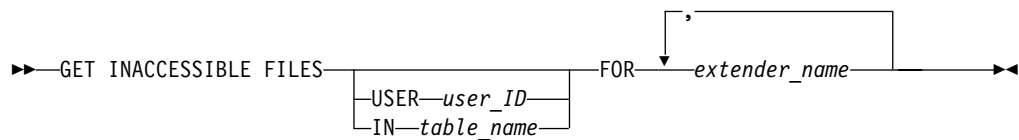
Image	Audio	Video
X	X	X

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 server.

Authorization

SYSADM, or SELECT privilege on enabled columns in all searched user tables and associated administrative support tables

Command syntax



Command parameters

user_ID

The qualifier of the tables in the current database server whose inaccessible files you want to list.

table_name

The name of the table in the current database server 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 server, 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
```

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 server 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

GET INACCESSIBLE FILES

that qualifier. If you specify neither, the command lists inaccessible files for all the tables in the current database server.

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.
- The manual cataloging indicator.

Authorization

- SYSADM
- DBADM on the database
- User ID of MMDBSYS or a user with a secondary authorization ID of MMDBSYS; MMDBSYS ID has SELECT privilege on the user table
- SELECT privilege on the user table and SELECT privilege on the QBIC catalog tables

Command syntax

►► GET QBIC CATALOG INFO ◀◀

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

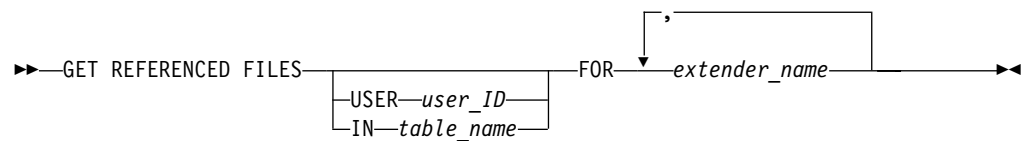
List all media files and the column names that reference them in a table, tables with a specific qualifier, or all the tables in the current database server.

Authorization

For all tables in the current database server, 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 server 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 server:

```
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 server before using this command.

GET REFERENCED FILES

If you do not specify any parameters, the command searches all the tables in the database server.

GRANT

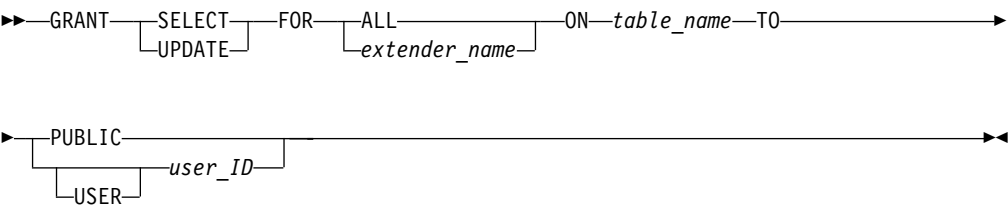
Image	Audio	Video
X	X	X

Grants privileges on administrative support tables to users, including privileges on administrative support tables for QBIC catalogs.

Authorization

SYSADM, DBADM

Command syntax



Command parameters

SELECT

Grants SELECT privilege on all administrative support tables associated with the user table for the specified extenders. If the Image Extender is specified (DB2Image), also grants SELECT privilege on the administrative support tables for the QBIC catalogs associated with the user table.

UPDATE

If the Image Extender is specified (DB2Image), grants INSERT, UPDATE, and DELETE privileges on the QBIC catalog tables associated with the user table.

ALL Grants the specified privileges on administrative support tables associated with the user table for the Image, Audio, and Video Extenders.

extender_name
The name of an extender (DB2Image, DB2Audio, or DB2Video). The specified privileges are granted on administrative support tables associated with the specified extender.

extender_name
The name of the user table.

PUBLIC
Grants the specified privileges to all users.

user_ID
The user ID to which the privileges will be granted.

Examples

Grant SELECT privilege on the administrative support tables for the Image Extender associated with the employee table, including the administrative support tables for the QBIC catalogs. Grant the privilege to user ID ajones:

GRANT

```
grant select for db2image on employee to user ajones
```

Usage notes

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

►►—OPEN QBIC CATALOG—*table_name*—*column_name*—————►◄

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

QUIT

Image	Audio	Video
X	X	X

Shuts down the db2ext command-line processor for command entry in interactive mode. For OS/390 clients, the connection to DB2 is dropped. For workstation clients, 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

►►—QUIT—◄◄

Command parameters

None.

Examples

Shut down the command-line interface for interactive mode:

```
quit
```

Usage notes

For OS/390 clients, QUIT drops the connection to DB2. For workstation clients, QUIT maintains the connection to DB2.

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

```
►►—REMOVE QBIC FEATURE—feature_name—————►◄
```

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.

REVOKE

REVOKE

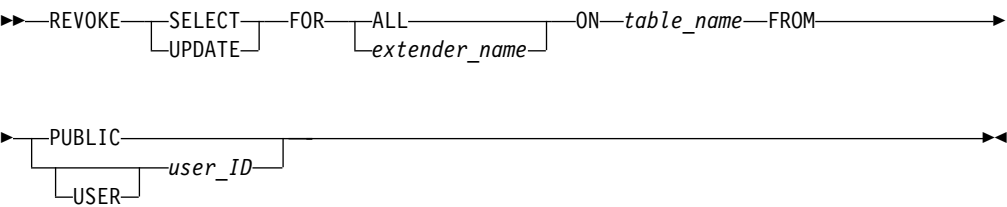
Image	Audio	Video
X	X	X

Revokes privileges on administrative support tables to users, including privileges on administrative support tables for QBIC catalogs.

Authorization

SYSADM, DBADM

Command syntax



Command parameters

SELECT

Revokes SELECT privilege on all administrative support tables associated with the user table for the specified extenders. If the Image Extender is specified (DB2Image), also revokes SELECT privilege on the QBIC catalog tables associated with the user table.

UPDATE

Revokes INSERT, UPDATE, and DELETE privileges on the administrative support tables for the QBIC catalogs associated with the user table.

ALL Revokes the specified privileges on administrative support tables associated with the user table for the Image, Audio, and Video Extenders.

extender_name

The name of an extender (DB2Image, DB2Audio, or DB2Video). The specified privileges are revoked on administrative support tables associated with the specified extender.

extender_name

The name of the user table.

PUBLIC

Revokes the specified privileges for all users.

user_ID

The user ID to which the privileges will be revoked.

Examples

Revoke SELECT privilege on the administrative support tables for the Image Extender associated with the employee table, including the administrative support tables for the QBIC catalogs. Revoke the privilege for user ID ajones:

```
revoke select for db2image on employee from user ajones
```

Usage notes

TERMINATE

TERMINATE

Image	Audio	Video
X	X	X

Shuts down the db2ext command-line processor and drops the connection to DB2.

Authorization

None

Command syntax

►►—TERMINATE—◄◄

Command parameters

None.

Examples

Shut down the db2ext command-line processor:

quit

Usage notes

TERMINATE drops the connection to the database server.

Chapter 16. 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 Application Programming and SQL 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 *ODBC Guide and Reference*.)

An SQLCODE value of 0 means that the statement ran successfully (with possible warning conditions). A positive SQLCODE value means that the statement was successfully run 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 382 for a description of each SQLSTATE code returned by the DB2 extenders.

Handling UDF codes

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 start DB2 extender UDFs.

See “SQLSTATE codes” for the SQLSTATE values and the message number of associated messages that can be returned by the extenders. See “Messages” on page 386 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 14. Application programming interfaces” on page 215 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 DBxGetError 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 DBxGetError API returns the SQL error code and associated message for the last DB2 extender API that encountered an error. See *DB2 Messages and Codes* for information about SQL error codes. See “Messages” on page 386 for information about each message that can be returned by the DBxGetError 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 13 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 386 for information about each message.

Table 13. 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

Table 13. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
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
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

SQLSTATEs

Table 13. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38623	DMB0208E DMB0131E	MMDB_SQLSTATE_PREPARE_FAIL Error occurred while preparing an SQL statement
38624	DMB0208E DMB0133E DMB0172E	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
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 occurred when an update UDF was invoked without referencing a table
38674	DMB0361E	MMDB_SQLSTATE_NOT_ENABLED Error occurred 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

Table 13. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38678	DMB0089E DMB0208E	MMDB_SQLSTATE_BASE_INTERNAL_ERROR Internal error in common layer
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_THUMBNAI_FILE_FAIL Cannot open thumbnail file
38696	DMB0135E	MMDB_SQLSTATE_READ_THUMBNAI_FILE_FAIL Cannot read thumbnail file
38697	DMB0207E	MMDB_SQLSTATE_OVERWRITE_NOT_ALLOWED The overwrite operation could not be performed
38699	DMB0171E	MMDB_SQLSTATE_QUERY_NAME_NOT_FOUND A query with that name was not found
38700		MMDB_SQLSTATE_NO_MANAGEBLOB
38701		MMDB_SQLSTATE_UDFLOCATOR_FAIL
38702		MMDB_SQLSTATE_SQL_FAIL
38703		MMDB_SQLSTATE_INVALID_UPDATE
38704		MMDB_SQLSTATE_NOT_ENABLED
38705	DMB0366E DMB0382E	MMDB_SQLSTATE_QBIC_QUERY_FAIL_TO_BUILD A failure occurred in building the query

SQLSTATEs

Table 13. SQLSTATE codes and associated message numbers (continued)

SQLSTATE	Message No.	Description
38706	DMB0205E	MMDB_SQLSTATE_QBIC_TABLE_COLUMN_PAIR_NOT_VALID A failure occurred when trying to access a QBIC catalog. Either an image handle wasn't found in the catalog, or the combination of the table name and column name does not have a catalog.
38707	DMB0383E	MMDB_SQLSTATE_QBIC_QUERY_EXECUTE_FAILED A failure occurred in running the query
38708		MMDB_SQLSTATE_QBIC_UNKNOWN_ERROR An unknown failure occurred in QBIC
38709	DMB0208E	MMDB_COPY_FILE_TO_LOCATOR_FAILURE A failure occurred in copying a file to a LOB locator
38710	DMB0534E	MMDB_SQLSTATE_QBIC_UNSUPPORTED_UDF The UDF is not supported.

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 that 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 that is 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 that is 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 that is shown in the message, so it cannot be enabled for that extender.

Action: Make sure that 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 referred to 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 that are 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 that are 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 referred to in table "<table-name>" for extender "<extender-name>".

Cause: The message displays the number of external media files that are referred to 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 referred to by user tables with a specific schema name.

Action: No action is required.

Messages

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 referred to 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:

- referred to 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 referred to 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 that 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 that 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 that are 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 that are 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 that the column is enabled for the extender that is 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 that the database exists and is enabled for the extender. Make sure that you are authorized to disable the database.

Messages

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 that the table exists and is enabled for the extender. Make sure that 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 that is 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 damaged, or it is not a media file.

Action: Verify that the file is a media file and is not damaged.

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 that 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 that is 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 referred to 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 that is 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 that is 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 damaged or erased.

Action: Check the name and verify the existence of the metadata table. If the metadata tables were accidentally erased or damaged, disable and then reenab 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 damaged.

Action: Verify your application to make sure that the extender handle values are not changed.

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.

Messages

DMB0076E Can't open import file.

Cause: The import file is either missing or damaged.

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 damaged 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 that 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 that 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 that is 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 that is being updated is different from the extender for which the column is enabled.

Action: Make sure that 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 that is 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 that is 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 that the file is a MPEG1 file.

DMB0106E AVI header invalid.

Cause: The header of the AVI file is missing or corrupt.

Action: Verify that the file is an AVI file.

Messages

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 that the environment variables are set properly, as described in "Appendix A. Setting environment variables for DB2 extenders" on page 409.

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 that the environment variables are set properly, as described in "Appendix A. Setting environment variables for DB2 extenders" on page 409.

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 that the environment variables are set properly, as described in "Appendix A. Setting environment variables for DB2 extenders" on page 409.

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 that the environment variables are set properly, as described in "Appendix A. Setting environment variables for DB2 extenders" on page 409.

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 409.

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 that the environment variables are set properly, as described in "Appendix A. Setting environment variables for DB2 extenders" on page 409.

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 that 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 that the environment variables are set properly, as described in "Appendix A. Setting environment variables for DB2 extenders" on page 409.

DMB0118E Can't open temporary file.

Cause: The temporary file might have been overwritten or damaged.

Action: Make sure that the environment variables are set properly, as described in "Appendix A. Setting environment variables for DB2 extenders" on page 409.

DMB0119I The dmbsrv server is starting for "<name>" with "<count>" connections.

Cause: The message indicates how many connections are made when the server starts.

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 that 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 are made when the server starts.

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 damaged.

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.

Messages

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: To add the feature to the query before calling other APIs that access the feature, use the QbQueryAddFeature API.

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 referred to 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 run 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 that 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 that are 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.

Messages

DMB0174E A system error occurred: "<error>".

Cause: An internal IBM error occurred.

Action: Follow the instructions that are 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 damaged.

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 damaged.

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 that is 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 damaged.

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 referred to 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 referred to 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 that 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.

Messages

Action: Verify that 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 that 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 that you issue the command db2ext terminate to end 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 not valid.

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.

Messages

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.

DMB0390E The command "<command-name>" or API "<api-name>" is valid only if the application is connected to a DB2 "<server-type>" server.

Cause: The application is connected to a type of DB2 server that the API or command does not support.

Action: Use the proper API or command for the type of DB2 server to which the application is connected.

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 started 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 started from a DB2 UDB client.

Action: Do not issue this command in the current client/server configuration.

DMB0395E This command cannot be run from a DB2 for OS/390 client.

Cause: DB2 extender client code supports administrative APIs and commands for a DB2 for OS/390 server, but not vice versa. DB2 extender commands that run in a DB2 UDB server (either Extended Enterprise Edition or non-Extended Enterprise Edition) are not supported by DB2 for OS/390.

Action: Do not issue this command in the current client/server configuration.

DMB0396I The current server is disabled for the "<extender-name>" extender.

Cause: The disable operation was successful.

Action: No action is required.

DMB0397E The current server cannot be disabled for the "<extender_name>" extender. RC = <code>

Cause: The database server does not exist or is not enabled for the extender, or you are not authorized to disable the database server.

Action: Make sure that the database server exists and is enabled for the extender. Make sure that you are authorized to disable the database server.

DMB0398I The current server is enabled for "<count>" extenders.

Cause: The message lists the number of DB2 extenders the current database server is enabled for.

Action: No action is required.

DMB0399E The current server cannot be enabled for the "<extender-name>" extender. RC = <code>

Cause: Either the database server does not exist, or you are not authorized to enable the database server.

Action: Make sure the database server exists and that you are authorized to enable the database server.

DMB0403E This command cannot be run with a DB2 OS/390 Server.

Cause: This command is acceptable if the db2ext command-lien processor is connected to a DB2 UDB server, but it cannot be run with a DB2 for OS/390 server.

Action: Do not use this CLP command under the current client/server configuration.

DMB0405E The specified tablespace is not defined on the "MMDBSYS" database.

Cause: The tablespace passed as input to the API or command is not defined on the "MMDBSYS" database.

Action: Use tablespaces defined on the "MMDBSYS" database as input to the DBxEnableServer() API and the ENABLE SERVER command.

DMB0406E The required tablespace specification is missing.

Cause: The required tablespace specification was not given with the API or command.

Action: Specify tablespaces that are defined on the "MMDBSYS" database as input when calling this API or command.

DMB0459E -a option is only applicable if you are a DB2 OS/390 client.

Cause: The -a option is used to specify an OS/390 subsystem, however it was issued from a DB2 UDB client.

Action: Do not specify the -a option from a DB2 UDB client.

DMB0463E The command "<command-name>" failed. The file "<file-name>" in the redist directory on the instance owning machine contains more information.

Cause: The specified command was not successfully run. See the file for more information.

Action: Make the necessary corrections and try again.

DMB0464E UDF error on node "<node-number>".

Cause: The UDF was not successfully run on the specified node.

Action: Make the necessary corrections and try again.

DMB0465E Error on node "<node-number>".

Cause: An error occurred on the specified node number.

Action: Make the necessary corrections and try again.

DMB0466E One or more nodes are not responding.

Cause: One or more nodes are currently not responding.

Action: Try again later.

DMB0467E "<variable-name>" is not a valid DB2 Extender variable.

Cause: The specified variable is not acceptable to the DB2 extender.

Action: Correct the variable name and try again.

DMB0468E The instance profile "<file-name>" is not defined.

Cause: The instance profile in the specified file name is not defined.

Action: Define the instance profile and try again.

DMB0469I The current server is enabled for "<extender-name>" extenders.

Cause: The current database server is enabled for the specified extender.

Action: No action is required.

Messages

DMB0470I **Attach to "<subsystem-name>".**

Cause: The attachment to the specified subsystem is successful.

Action: No action is required.

DMB0471I **Attach closed.**

Cause: The attachment to the specified subsystem is successfully closed.

Action: No action is required.

DMB0472I **The required WLM ENVIRONMENT specification is missing.**

Cause: An expected WLM environment specification is missing.

Action: Add the WLM environment specification and try again.

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 at an OS/390 server to:

- Start tracing
- Stop tracing
- Reformat trace information to make it more readable

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.

The DB2 extenders append the appropriate uid to the specified *path* to generate a file name for the trace file. The uid is based on the external security specification that is made when the database server is enabled. If EXTERNAL SECURITY DB2 is specified, the uid for the pertinent WLM environments is used. If EXTERNAL SECURITY USER is specified, the uid is the primary authorization ID of the user or application that started the pertinent process. For further information about external security specifications, see "Specifying external security" on page 42.

For example, the following command starts tracing:

`dmbtrc on /tmp/trace`

If EXTERNAL SECURITY DB2 was specified for the database server, and the uid of the pertinent WLM environments is 99, the extenders will record trace information in file `/tmp/trace.99`. If EXTERNAL security user was specified for the database server, and the uid of the user or program that started the pertinent process is 1, the extenders will record trace information in file `/tmp/trace.1`. Trace data is recorded in a trace file only if the pertinent uid and gid can be created for the specified directory, and if the uid and gid have permission to write to the file.

DB2MMTRACE must be set: Before tracing can begin, the DB2MMTRACE environment variable must be set to the file name in the WLM startup procedure. You must also set the value of the environment variable DB2MMTRACE before you first use the DMBTRC ON command. The file name that you specify for DB2MMTRACE should be the same as the file name specified for DB2MMTRACE in the WLM startup procedure. When you enter the DMBTRC ON command, the extenders create the file specified in DB2MMTRACE, and enter into the file the name of the trace file path specified in the DMBTRC ON command. For performance reasons, it is best to set DB2MMTRACE to the name of a file that currently does not exist. For further information about setting environment variables in the WLM environment startup procedure, see “Setting environment variables in OS/390” on page 410.

You need file permissions: To run the DMBTRC command, you need create, read, write, and delete permission on the file identified by DB2MMTRACE.

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. You can specify multiple input files, but only one output file (the last file specified is assumed to be the output file).

For example, the following command reformats trace information:

```
dmbtrc format /tmp/extender/trace.1 /tmp/fmt/trace.1.fmt
```

OS/390 Open Edition supports wildcard specification in file names. For example, the following command in OS/390 Open Edition reformats trace information in multiple input files, and merges it into a single output file.

```
dmbtrc format /tmp/extender/trace.* /tmp/fmt/trace.fmt
```

Tracing

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 that are 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 a file system such as HFS, or in AIX or Solaris, a relative file name is any file name that does not begin with a slash. In Windows, a relative file name is any file name that does not begin with a drive letter that is 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 14 lists and describes environment variables that you can set for use by the Image, Audio, and Video Extenders in resolving file names.

Table 14. 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.
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.

Environment variables

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.

Setting environment variables

You can set environment variables in OS/390, AIX, Solaris, and Windows.

Setting environment variables in OS/390

In OS/390, environment variables are specified in a DSN in the DMBENVAR DD card in the WLM managed startup proc, for example, DMBWLM1. The following is a sample DMBWLM1 procedure:

```
//V61AWLM1 PROC DB2SSN=V61A,NUMTCB=18,APPLENV=DMBWLM1
//TCBNUM1 EXEC PGM=DSNX9WLM,TIME=1440,
//          PARM='&DB2SSN,&NUMTCB,&APPLENV',REGION=0M
//STEPLIB DD DSN=DSN610.SDSNLOAD,DISP=SHR <== DB2 Load Library
//          DD DSN=DMB.V61.LOADLIB,DISP=SHR <== DB2EXT Load library
//CEEDUMP DD SYSOUT=H
//SYSPRINT DD SYSOUT=H
//SYSDUMP DD SYSOUT=H
//DMBENVAR DD DISP=SHR,DSN=&HQL.DMB.ENVAR <= DB2EXT ENV Variable File
//*DSNAOTRC DD DISP=SHR,DSN=&HQL.DMB.DSNAOTRC <= DB2 CLI Trace
```

The following is a sample of the contents of global environment variables defined in file DMB.ENVAR:

```
DB2MMPATH=/usr/lpp/db2ext_07_01_00/samples:/tmp
DB2MMTEMP=/tmp
DB2MMSTORE=/tmp
DB2MMEEXPORT=/tmp
```

Setting environment variables in AIX and Solaris clients

In AIX and Solaris, the environment variables are specified in C shell, Korn shell, and Bourne shell scripts.

The client environment variables are set as follows when you install the DB2 Extenders in an AIX 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 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/db2ext/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/db2ext/samples:/media
export DB2MMPATH
```

Environment variables

```
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/db2ext/samples:/media  
export DB2IMAGEPATH=/employee/pictures:/images  
export DB2AUDIOSTORE=/employee/sounds
```

Setting environment variables in Windows clients

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.

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 MVS partitioned data set of the target library, and in the SAMPLES Open Edition directory when you install the DB2 extenders. The image, audio, and video files are also installed in the SAMPLES Open Edition directory when you install the DB2 extenders. For installation, you need to ensure that the extenders environment variable DB2MMPATH is set to point to the SAMPLES directory.

Sample programs

A number of files comprise the sample programs for DB2 Extenders.

Open Edition	MVS	Description
enable.c	ENABLE	Enables a database server for the Audio, Image, and Video Extenders, creates a table, and enables the table and its columns.
populate.c	POPULATE	Imports data into the table.
query.c	QUERY	Queries the data in the table.
api.c	API	Queries the database server using extender APIs.
handle.c	HANDLE	Demonstrates the use of handles in UDFs and how to make where clause comparisons in SELECT statements.
qbcatdmo.c	QBCATDMO	Creates a QBIC catalog and catalogs a column of images into the catalog
qbicdemo.c	QBICDEMO	Queries a QBIC catalog
color.c	COLOR	Makes color table declarations for qbicdemo.c
utility.c	UTILITY	Utility routines.
utility.h	UTILITY	Header file for utility routines
makefile.aix		Makefile to build the programs in AIX
makefile.iva		Makefile to build the programs in Windows NT (or later) using IBM VisualAge C++
makefile.mvc		Makefile to build the programs using Microsoft Visual C++
makefile.sun		Makefile to build the programs in Solaris

Sample programs

Executable files are provided for the following sample programs. The sample programs are intended to be run in the order that is shown.

1. Enable
2. Populate
3. Query
4. API
5. Handle
6. Qbcatdmo
7. Qbicdemo

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 starts the program's executable file). You will be prompted for the DB2 for OS/390 location name.

You can also build in TSO or UNIX environments your own executable files for the sample programs. To do that, you need to:

1. Copy the sample media files to a writable Open Edition directory (the DB2MMPATH environment variable should point to this directory).
2. **For TSO:**

Edit the sample JCL (see Figure 25 on page 415) to specify the locations on your system where the extenders, DB2, and the compiler are installed. In the sample JCL, the libraries (PDS) are allocated as:

DMB.V61.SAMPLE.C — sample C code
DMB.V61.SAMPLE.H — sample header files
DMB.V61.SAMPLE.OBJLIB — object library
DMB.V61.SAMPLE.PRELINK — prelink library
DMB.V61.SAMPLE.LOADLIB — load library

Submit the JCL to compile, prelink, and link edit the sample programs.

For UNIX:

Use the OEDIT command to edit the makefile.

3. Bind a sample plan (see Figure 26 on page 419). The default plan name is DMBACLI.
4. Set up the CLI.INI file (see Figure 27 on page 420).
5. For TSO, set STEPLIB concatenation (see Figure 28 on page 420).

For further information about installing and using the sample programs, see the *Program Directory for IBM Database 2 Universal Database Server for OS/390, Volume 1 of 8*.

Step 1) Compile the sample programs
 Example of compile sample JCL

```
//DMBSAMC JOB CLASS=A,TIME=(15,59),
//      MSGCLASS=H,MSGLEVEL=(1,1),USER=&SYSUID,REGION=0K,
//      NOTIFY=&SYSUID,PASSWORD=&SYSUID
//*ROUTE PRINT hostname.userid
/*-----
//DMBC  PROC LNGPRFX='CBC',
//      CLANG='EDCMSGE',      USED IN THIS RELEASE. KEPT FOR COM
//      CXXLANG='CBCMSG',    USED IN THIS RELEASE. KEPT FOR COM
//      CREGSIZ='16M',
//      CPARAM=,              < COMPILER OPTIONS
//      CPARAM2=,            < COMPILER OPTIONS
//      CPARAM3=,            < COMPILER OPTIONS
//      LIBPRFX='SYS1',
//      TUNIT='VIO',
//      OPT=OPTC,             < AIV DFLT OPT FILE
//DCB80='(RECFM=FB,LRECL=80,BLKSIZE=3200)',      FOR LRECL 80
//DCB3200='(RECFM=FB,LRECL=3200,BLKSIZE=12800)'  FOR LRECL 3200
/*-----
/* SAMPLE
/*-----
//API      EXEC  DMBC,M=API,CPARM=NOEXPORTALL
//ENABLE   EXEC  DMBC,M=ENABLE,CPARM=NOEXPORTALL
//HANDLE   EXEC  DMBC,M=HANDLE,CPARM=NOEXPORTALL
//QUERY    EXEC  DMBC,M=QUERY,CPARM=NOEXPORTALL
//UTILITY  EXEC  DMBC,M=UTILITY
```

Compile option file as:

```
SEARCH('DSN610.SDSNC.+','DMB.V61.SAMPLE.+','DMB.V61A.MACS')
SEARCH('SYS1.SCEEH.+','CBC.SCLBH.+')
LSEARCH('DSN610.SDSNC.+')
DEF(UEDEBUG,DMB_EXTENDERS,DMB_MVS,DMB_UNIX,DMB_NO_COLLECTIONS,DB2MVS)
DEF(_OPEN_THREADS,_XOPEN_SOURCE_EXTENDED,_OPEN_DEFAULT,NO_X11)
DEF(errno=(*_errno()),NO_DB2)
DEF(DMB_UNIX_DIRECTORY="/opt/IBMdb2ex/V6.0/")
NOMAR,NONSEQ,LOCALE(POSIX),LANGVLV(ANSI)
OBJECT,LONGNAME,RENT
DLL,EXPORTALL
NOAGGR,NOXREF,NOLIST,SOURCE,NOEXPMAC,NOSHOWINC,SSCOMM,NONESTINC(*)
TEST(ALL)
```

Figure 25. Sample JCL (Part 1 of 5)

Sample programs

```
Step 2) Prelink and Linkedit
//DMBSAMPK JOB CLASS=A,TIME=(,15),
//      MSGCLASS=H,MSGLEVEL=(1,1),USER=&SYSUID,REGION=0K
//*      PASSWORD=&SYSUID
//*ROUTE PRINT hostname.userid
//*-----
//* PRELINK: SAMPLES
//*-----
//* PRELINK INLINE PROC AND PARMS
//*-----
//DMBSAMPK PROC LIBPRFX='SYS1',LNGPRFX='CBC',M=
//PRELINK EXEC PGM=EDCPRLK,
//  PARM='POSIX(OFF)/OE,MEMORY,DUP,NOER,MAP,NOUPCASE,NONCAL,OMVS'
//STEPLIB DD DISP=SHR,DSN=&LIBPRFX..SCEERUN          <== INPUT CEE/EC
//SCEE0BJ DD DISP=SHR,DSN=&LIBPRFX..SCEE0BJ          <== INPUT CEE/EC
//SCEECPD DD DISP=SHR,DSN=&LIBPRFX..SCEECPD          <== INPUT CEE/EC
//SCLBCPD DD DISP=SHR,DSN=&LNGPRFX..SCLBCPD          <== INPUT COMPILE
//SYSMSGD DD DISP=SHR,DSN=&LIBPRFX..SCEEMSGD(EDCPMSGD)
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(10,10))
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//*
//CBCLIB DD DISP=SHR,DSN=&LNGPRFX..SCLBSID          <== INPUT CBC
//DB2XLIB DD DISP=SHR,DSN=DMB.DSN610.SDSNMACS.EXP   <== INPUT DB2 EXP
//OBJLIB DD DISP=SHR,DSN=DMB.V61.SAMPLE.OBJLIB      <== INPUT OBJS
//EXPLIB DD DISP=SHR,DSN=DMB.V61A.EXP               <== INPUT EXP
//SYSMOD DD DISP=SHR,DSN=DMB.V61.SAMPLE.PRELINK(&M) <== OUTPUT DLL
//SYSDEFSD DD DISP=SHR,DSN=DMB.V61.SAMPLE.EXP(&M)  <== OUTPUT EXP
//SYSIN DD DUMMY
//DMBSAMPK PEND
//*-----
//* SAMPLE: API
//*-----
//API EXEC DMBSAMPK,M=API
//PRELINK.SYSIN DD *
INCLUDE OBJLIB(API)
INCLUDE OBJLIB(UTILITY)
INCLUDE DB2XLIB(DSNAOCLI)
INCLUDE EXPLIB(DMBAUDIO)
INCLUDE EXPLIB(DMBIMAGE)
INCLUDE EXPLIB(DMBVIDEO)
LIBRARY SCEE0BJ
LIBRARY SCEECPD
```

Figure 25. Sample JCL (Part 2 of 5)


```

/*-----
/* SAMPLE: ENABLE
/*-----
//ENABLE EXEC DMBSAMPK,M=ENABLE
//PRELINK.SYSIN DD *
INCLUDE OBJLIB(ENABLE)
INCLUDE OBJLIB(UTILITY)
INCLUDE DB2XLIB(DSNAOCLI)
INCLUDE EXPLIB(DMBAUDIO)
INCLUDE EXPLIB(DMBIMAGE)
INCLUDE EXPLIB(DMBVIDEO)
LIBRARY SCEEOBJ
LIBRARY SCEECPP
/*-----
/* SAMPLE: HANDLE
/*-----
//HANDLE EXEC DMBSAMPK,M=HANDLE
//PRELINK.SYSIN DD *
INCLUDE OBJLIB(HANDLE)
INCLUDE OBJLIB(UTILITY)
INCLUDE DB2XLIB(DSNAOCLI)
INCLUDE EXPLIB(DMBAUDIO)
INCLUDE EXPLIB(DMBIMAGE)
INCLUDE EXPLIB(DMBVIDEO)
LIBRARY SCEEOBJ
LIBRARY SCEECPP
/*-----
/* SAMPLE: POPULATE
/*-----
//POPULATE EXEC DMBSAMPK,M=POPULATE
//PRELINK.SYSIN DD *
INCLUDE OBJLIB(POPULATE)
INCLUDE OBJLIB(UTILITY)
INCLUDE DB2XLIB(DSNAOCLI)
INCLUDE EXPLIB(DMBAUDIO)
INCLUDE EXPLIB(DMBIMAGE)
INCLUDE EXPLIB(DMBVIDEO)
LIBRARY SCEEOBJ
LIBRARY SCEECPP

```

Figure 25. Sample JCL (Part 3 of 5)

Sample programs

```
/*-----  
/* SAMPLE: QUERY  
/*-----  
/*QUERY EXEC DMBSAMPK,M=QUERY  
/*PRELINK.SYSIN DD *  
INCLUDE OBJLIB(QUERY)  
INCLUDE OBJLIB(UTILITY)  
INCLUDE DB2XLIB(DSNAOCLI)  
INCLUDE EXPLIB(DMBAUDIO)  
INCLUDE EXPLIB(DMBIMAGE)  
INCLUDE EXPLIB(DMBVIDEO)  
LIBRARY SCEEOBJ  
LIBRARY SCEECPP  
Example of Linkedit JCL:  
//DMBSAMLK JOB CLASS=A,TIME=(,15),  
// MSGCLASS=H,MSGLEVEL=(1,1),USER=&SYSUID,REGION=0K  
/* PASSWORD=&SYSUID  
/*ROUTE PRINT STLVM6.xxxx  
/*-----  
/* LINKEDIT: ALL CLIENT FUNCS (COMMAND AND *CLT)  
/*-----  
//DMBCLTLK PROC LIBPRFX='SYS1'  
//LKED EXEC PGM=LINKEDIT,REGION=16M,  
// PARM=('AMODE=31,RMODE=ANY,LIST','REUS=RENT,CALL,CASE=MIXED',  
// 'DYNAM=DLL,MAP')  
//SYSLIB DD DISP=SHR,DSNAME=&LIBPRFX..SCEELKEX  
// DD DISP=SHR,DSNAME=&LIBPRFX..SCEELKED  
// DD DISP=SHR,DSNAME=&LIBPRFX..CSSLIB  
//DB2LIB DD DISP=SHR,DSNAME=DMB.DSN610.SDSNLOAD <== INPUT DSNHLI  
//PRELIB DD DISP=SHR,DSN=DMB.V61.SAMPLE.PRELINK <== INPUT PRELINK  
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(10,10))  
//SYSPRINT DD SYSOUT=*  
//SYSLOAD DD DISP=SHR,DSN=DMB.V61.SAMPLE.LOADLIB <== OUT LOAD MODS  
//SYSLIN DD DUMMY  
//DMBCLTLK PEND
```

Figure 25. Sample JCL (Part 4 of 5)

```
/*-----  
/* COMMAND AND *CLT LINKEDIT LIST  
/*INCLUDE DB2LIB(DSNTIAR)  
/*INCLUDE DB2LIB(DSNALI)  
/*-----  
//DMBCLT EXEC DMBCLTLK  
//LKED.SYSLIN DD *  
INCLUDE PRELIB(API)  
NAME API(R)  
  
INCLUDE PRELIB(ENABLE)  
NAME ENABLE(R)  
  
INCLUDE PRELIB(HANDLE)  
NAME HANDLE(R)  
  
INCLUDE PRELIB(POPULATE)  
NAME POPULATE(R)  
  
INCLUDE PRELIB(QUERY)  
NAME QUERY(R)  
/*
```

Figure 25. Sample JCL (Part 5 of 5)

```

//BINDALL JOB CLASS=A,
//          MSGCLASS=H,MSGLEVEL=(1,1),REGION=0K,TIME=NOLIMIT,
//          USER=SYSADM,PASSWORD=SYSADM,NOTIFY=&SYSUID
//*ROUTE PRINT hostname.userid
//*****/
//JOB LIB DD DISP=SHR,DSN=DB2A.SDSNLOAD
//*
//          SET DMBFQP=DMB.V61A
//*
//BIND CLI EXEC PGM=IKJEFT01,DYNAMNBR=20
//DBRMLIB DD DISP=SHR,
//          DSN=REDEM.DBRMLIB.DATA
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(V61A)

BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLICS) ISOLATION(CS)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLINC) ISOLATION(NC)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLIRR) ISOLATION(RR)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLIRS) ISOLATION(RS)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLIUR) ISOLATION(UR)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLIC1)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLIC2)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLIF4)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLIMS)
BIND PACKAGE (DSNAOCLI) MEMBER(DSNCLIQR)

BIND PLAN(DMBACLI)
PKLIST(DSNAOCLI.DSNCLICS -
      DSNAOCLI.DSNCLINC -
      DSNAOCLI.DSNCLIRR -
      DSNAOCLI.DSNCLIRS -
      DSNAOCLI.DSNCLIUR -
      DSNAOCLI.DSNCLIC1 -
      DSNAOCLI.DSNCLIC2 -
      DSNAOCLI.DSNCLIF4 -
      DSNAOCLI.DSNCLIMS -
      DSNAOCLI.DSNCLIQR -
      MMDBSYS_CLIENT.* -
      MMDBSYS_RUN.* )
END
/*

```

Figure 26. Sample Bind

Sample programs

Under TSO, a DD name of DSNAOINI must be allocated for the DB2 CLI initialization processing.

```
alloc dd(dsnaoini) da('&HQL.cli.ini6') shr
```

Example of CLI.INI file(&HQL.cli.ini6):

```
; Common stanza
[COMMON]
MVSDEFAULTSSID=V61A
; Subsystem stanza for V61A subsystem using DMBACLI plan
;
[V61A]
PLANNAME=DMBACLI
; Data source stanza for STLEC1
[STLEC1]
;
```

Figure 27. Setting up the CLI.INI file

Under TSO, use TSOLIB command to setup the DB2 LOADLIB and DB2 Extenders LOADLIB.

Example of TSO Libraries concatenation

```
tsolib act dsn('DMB.V61.LOADLIB','DSN610.SDSNLOAD')
```

where DMB.V61.LOADLIB is DB2 Extender load library, and DSN610.SDSNLOAD is DB2 load library.

Figure 28. Sample STEPLIB concatenation

Sample image, audio, and video files

The sample image, audio, and video files that are 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

Appendix C. 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 for OS/390 and develop programs for use with DB2 extenders for OS/390. This publication documents General-use Programming Interface and Associated Guidance Information provided by DB2 extenders for OS/390.

General-use programming interfaces allow you to write programs that obtain the services of DB2 extenders for OS/390. You may copy the DB2 extenders for OS/390 run-time feature needed for the application you develop onto client or server machines. To install the run-time feature onto a client machine, see the installation instructions provided in the README.TXT file for your operating system on the DB2 extenders for OS/390 clients CD-ROM. To install the run-time feature onto a server machine, see the installation instructions provided in the extender sections of the Program Directory for DB2 for OS/390 V6.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX	DB2 Universal Database	OS/390
DB2	IBM	QBIC
DB2 Extenders	OS/2	

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 table. One of the tables that are 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.

API. See *application programming interface*.

application programming interface (API).

(1) A functional interface supplied by the operating system or by a separately orderable licensed program. An API allows an application program that is written in a high-level language to use specific data or functions of the operating system or the licensed programs.

(2) In DB2, a function within the interface, for example, the get error message API.

(3) The DB2 extenders provide APIs for requesting user-defined functions, administrative operations, and display operations.

audio. Pertaining to the portion of recorded information that can be heard.

average color. A measurement of color that is computed as an average of the color values that are 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.

DB2 extender. One of a group of programs that you use store and retrieve data types beyond the traditional numeric and character data, such as image, audio, and video data, and complex documents.

directionality. An attribute of *texture* that describes whether the image has a favored direction (like grass) or is like a smooth object (like glass).

distinct type. See *user-defined type*.

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.

environment variable. A variable used to describe the operating environment for the DB2 extenders and to provide defaults for values for the environment.

extender. See *DB2 extender*.

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 that is 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 that is stored in a user table with information about the object that is 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 referred to 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.

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. See *administrative support table*.

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.

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 that is 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

score. A calculated value that reflects how similar the feature values are to those that are 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.

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.

UDF. See *user-defined function*.

UDT. See *user-defined type*.

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 that is 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.

Index

A

- access privileges 32
- ADD QBIC FEATURE command 111, 353
- administration commands on client 351
 - ADD QBIC FEATURE 353
 - CATALOG QBIC COLUMN 354
 - CLOSE QBIC CATALOG 355
 - CREATE QBIC CATALOG 356
 - DELETE QBIC CATALOG 358
 - DISABLE COLUMN 359
 - DISABLE SERVER 360
 - DISABLE TABLE 361
 - ENABLE COLUMN 362
 - ENABLE SERVER 363
 - ENABLE TABLE 365
 - GET EXTENDER STATUS 367
 - GET INACCESSIBLE FILES 368
 - GET QBIC CATALOG INFO 370
 - GET REFERENCED FILES 371
 - GRANT 373
 - OPEN QBIC CATALOG 375
 - QUIT 376
 - REMOVE QBIC FEATURE 377
 - REVOKE 378
 - TERMINATE 380
- administration task overview 37
- administrative support tables 14
 - description 14
 - security 32
- alignment value of audio or video 149
- AlignValue UDF 149
- allocation resources 69
- application programming interfaces (APIs) 215
 - DBaAdminGetInaccessibleFiles 216
 - DBaAdminGetReferencedFiles 218
 - DBaAdminIsFileReferenced 220
 - DBaDisableColumn 222
 - DBaDisableServer 224
 - DBaDisableTable 225
 - DBaEnableColumn 227
 - DBaEnableServer 229
 - DBaEnableTable 231
 - DBaGetError 233
 - DBaGetInaccessibleFiles 234
 - DBaGetReferencedFiles 236
 - DBaIsColumnEnabled 238
 - DBaIsFileReferenced 240
 - DBaIsServerEnabled 242
 - DBaIsTableEnabled 243
 - DBaPlay 245
 - DBaPrepareAttrs 247
 - DBiAdminGetInaccessibleFiles 248
 - DBiAdminGetReferencedFiles 250
 - DBiAdminIsFileReferenced 252
 - DBiBrowse 254
 - DBiDisableColumn 256
 - DBiDisableServer 258
 - DBiDisableTable 259
 - DBiEnableColumn 261
- application programming interfaces (APIs) 215 (*continued*)
 - DBiEnableServer 263
 - DBiEnableTable 265
 - DBiGetError 267
 - DBiGetInaccessibleFiles 268
 - DBiGetReferencedFiles 270
 - DBiIsColumnEnabled 272
 - DBiIsFileReferenced 274
 - DBiIsServerEnabled 276
 - DBiIsTableEnabled 277
 - DBiPrepareAttrs 278
 - DBvAdminGetInaccessibleFiles 279
 - DBvAdminGetReferencedFiles 281
 - DBvAdminIsFileReferenced 283
 - DBvDisableColumn 285
 - DBvDisableServer 287
 - DBvDisableTable 288
 - DBvEnableColumn 290
 - DBvEnableServer 292
 - DBvEnableTable 294
 - DBvGetError 296
 - DBvGetInaccessibleFiles 297
 - DBvGetReferencedFiles 299
 - DBvIsColumnEnabled 301
 - DBvIsFileReferenced 303
 - DBvIsServerEnabled 305
 - DBvIsTableEnabled 306
 - DBvPlay 307
 - DBvPrepareAttrs 309
 - QbAddFeature 310
 - QbCatalogColumn 312
 - QbCloseCatalog 314
 - QbCreateCatalog 315
 - QbDeleteCatalog 317
 - QbGetCatalogInfo 319
 - QbListFeatures 320
 - QbOpenCatalog 322
 - QbQueryAddFeature 324
 - QbQueryCreate 326
 - QbQueryDelete 327
 - QbQueryGetFeatureCount 328
 - QbQueryGetString 329
 - QbQueryListFeatures 330
 - QbQueryNameCreate 332
 - QbQueryNameDelete 334
 - QbQueryNameSearch 335
 - QbQueryRemoveFeature 337
 - QbQuerySearch 339
 - QbQuerySetFeatureData 341
 - QbQuerySetFeatureWeight 343
 - QbQueryStringSearch 344
 - QbReCatalogColumn 346
 - QbRemoveFeature 348
- aspect ratio of video 150
- AspectRatio UDF 150
- attributes, object 88
 - alignment value 149
 - aspect ratio 150
 - audio channels (number of) 189
 - bits per sample of audio 151
- attributes, object 88 (*continued*)
 - clock speed per quarter note 210
 - clock speed per second 211
 - colors in image (number of) 190
 - comment 153
 - compression format of video 155
 - data transfer rate of audio 152
 - data transfer rate of video 187
 - description 88
 - duration of audio or video 176
 - file name 177
 - format 180
 - frame rate of video 181
 - frames in video (number of) 191
 - height 184
 - import time 186
 - importer 185
 - number of audio channels 189
 - number of audio tracks 188
 - number of colors in image 190
 - number of frames in video 191
 - number of video tracks 192
 - playing time of audio or video 176
 - sampling rate of audio 206
 - size 207
 - throughput of audio 152
 - throughput of video 181, 187
 - time stored 186
 - time updated 213
 - track name, MIDI 179
 - track names, MIDI 183
 - track number of all MIDI instruments 182
 - track number of MIDI instrument 178
 - update time 213
 - updater 212
 - user ID of person who stored 185
 - user ID of person who updated 212
 - video tracks (number of) 192
 - width 214
- audio 3
 - alignment of 149
 - bits per sample 151
 - channels (number of) 189
 - clock speed, MIDI 210, 211
 - comment attribute 153
 - data transfer rate 152
 - duration 176
 - file name 177
 - format attribute 180
 - formats 71
 - identifying format for storage 79
 - identifying format for update 98
 - import time 186
 - importer 185
 - number of channels 189
 - number of tracks 188
 - playing 103
 - playing time 176
 - retrieving 84

- audio 3 (*continued*)
 - sampling rate 206
 - size 207
 - storing 73
 - throughput 152
 - time stored 186
 - time updated 213
 - track name, MIDI 179
 - track names, MIDI 183
 - track number of all MIDI
 - instruments 182
 - track number of MIDI
 - instrument 178
 - tracks in (number of) 188
 - update time 213
 - updater 212
 - updating 90
 - user ID of person who stored 185
 - user ID of person who updated 212
- Audio Extender 4
 - DBaAdminGetInaccessibleFiles
 - API 216
 - DBaAdminGetReferencedFiles
 - API 218
 - DBaAdminIsFileReferenced API 220
 - DBaDisableColumn API 222
 - DBaDisableServer API 224
 - DBaDisableTable API 225
 - DBaEnableColumn API 227
 - DBaEnableServer API 229
 - DBaEnableTable API 231
 - DBaGetError API 233
 - DBaGetInaccessibleFiles API 234
 - DBaGetReferencedFiles API 236
 - DBaIsColumnEnabled API 238
 - DBaIsFileReferenced API 240
 - DBaIsServerEnabled API 242
 - DBaIsTableEnabled API 243
 - DBaPlay API 245
 - overview 4
 - UDFs 145
 - UDTs 145
- authorization 35
- average color 16
 - description 16
 - feature name 111

B

- backup 35
- binary large object (BLOB) 11
 - description 11
 - security 32
 - storing an object as 78
 - updating 97
- bind file 67
- binding 67
- BitsPerSample UDF 151
- buffer, client 64
 - retrieving to with conversion 86
 - retrieving to without format
 - conversion 85
 - storing from 77
 - transmitting an object to or from 64
 - updating from 96
- BytesPerSec UDF 152

C

- Call Attachment Facility Open
 - function 69
- catalog (QBIC) 16
 - adding a feature to 111
 - closing 115
 - creating 108
 - deleting 115
 - description 16
 - managing 108
 - opening 110
 - recataloging an image in 114
 - removing a feature from 112
 - retrieving information about 112
- CATALOG QBIC COLUMN
 - command 114, 354
 - cataloging a column 114
 - recataloging an image 114
- channels, number of audio 189
- character large object (CLOB) 11
- CLI calls, preparing 67
- client buffer 64
 - retrieving to with conversion 86
 - retrieving to without format
 - conversion 85
 - storing from 77
 - transmitting an object to or from 64
 - updating from 96
- client file 65
 - retrieving to 86
 - storing from 77
 - transmitting an object to or from 65
 - updating from 96
- client/server platforms for DB2
 - extenders 9
- CLOB (character large object) 11
- CLOSE QBIC CATALOG command 115, 355
- coarseness 17
- codes, return 381
- colors, number of (in image) 190
- columns 48
 - disabling 49
 - enabling 48
- commands 351
 - ADD QBIC FEATURE 353
 - CATALOG QBIC COLUMN 354
 - CLOSE QBIC CATALOG 355
 - CREATE QBIC CATALOG 356
 - DELETE QBIC CATALOG 358
 - DISABLE COLUMN 359
 - DISABLE SERVER 360
 - DISABLE TABLE 361
 - ENABLE COLUMN 362
 - ENABLE SERVER 363
 - ENABLE TABLE 365
 - GET EXTENDER STATUS 367
 - GET INACCESSIBLE FILES 368
 - GET QBIC CATALOG INFO 370
 - GET REFERENCED FILES 371
 - GRANT 373
 - OPEN QBIC CATALOG 375
 - QUIT 376
 - REMOVE QBIC FEATURE 377
 - REVOKE 378
 - TERMINATE 380
- comment 83

- comment 83 (*continued*)
 - retrieving 90
 - storing 83
 - updating 101
- Comment UDF 153
- compatibility mode 32
- compiling 67
- compression format of video 155
- compression type 72
- CompressType UDF 155
- concepts 11
- Content UDF 156
- ContentA UDF 160
- contrast 17
- conversion options, image 72
- CREATE QBIC CATALOG
 - command 108, 356
- current path 13
- CURRENT PATH special register 13
- CURRENT SERVER special register 73

D

- data structures 14
 - administrative support table 14
 - handle 15
 - QBIC catalog 16
- data transfer rate of audio 152
- data transfer rate of video 187
- Database Request Module (DBRM) 67
- database servers 42
 - checking if enabled 51
 - enabling 42
- DB2 extender 3
 - codes 381, 382
 - concepts 11
 - data structures 14
 - family of 4
 - operating environments 9
 - overview 3
 - planning for 31
 - programming overview 59
 - retrieving objects using 71
 - return codes 381
 - run-time environment 4
 - sample media files 413
 - sample programs 413
 - scenario 19
 - security 32
 - Software Developers Kit (SDK) 4
 - SQLSTATE codes 382
 - storing objects using 71
 - tasks that can be performed with 60
 - trace facility 404
 - UDFs 145
 - UDTs 145
 - updating objects using 71
- DB2 ODBC environment 67
- DB2AUDIO data type 145
- DB2Audio UDF 162
- DB2AudioA UDF 164
- DB2AUDIOEXPORT environment
 - variable 409
- DB2AUDIOPATH environment
 - variable 409
- DB2AUDIOPLAYER environment
 - variable 104

DB2AUDIOSTORE environment variable 409

DB2AUDIOTEMP environment variable 409

db2ext command-line processor 4

DB2IMAGE data type 145

DB2Image UDF 167

DB2ImageA UDF 170

DB2IMAGEBROWSER environment variable 103

DB2IMAGEEXPORT environment variable 409

DB2IMAGEPATH environment variable 409

DB2IMAGESTORE environment variable 409

DB2IMAGETEMP environment variable 409

DB2VIDEO data type 145

DB2Video UDF 172

DB2VideoA UDF 174

DB2VIDEOEXPORT environment variable 409

DB2VIDEOPATH environment variable 409

DB2VIDEOPLAYER environment variable 104

DB2VIDEOSTORE environment variable 409

DB2VIDEOTEMP environment variable 409

DBaAdminGetInaccessibleFiles API 216

DBaAdminGetReferencedFiles API 218

DBaAdminIsFileReferenced API 220

DBaDisableColumn API 222

DBaDisableServer API 224

DBaDisableTable API 225

DBaEnableColumn API 227

DBaEnableServer API 229

DBaEnableTable API 231

DBaGetError API 233

DBaGetInaccessibleFiles API 234

DBaGetReferencedFiles API 236

DBaIsColumnEnabled API 238

DBaIsFileReferenced API 240

DBaIsServerEnabled API 242

DBaIsTableEnabled API 243

DBaPlay API 245

DBaPrepareAttrs API 247

DBCLOB (double-byte character large object) 11

DBiAdminGetInaccessibleFiles API 248

DBiAdminGetReferencedFiles API 250

DBiAdminIsFileReferenced API 252

DBiBrowse API 254

DBiDisableColumn API 256

DBiDisableServer API 258

DBiDisableTable API 259

DBiEnableColumn API 261

DBiEnableServer API 263

DBiEnableTable API 265

DBiGetError API 267

DBiGetInaccessibleFiles API 268

DBiGetReferencedFiles API 270

DBiIsColumnEnabled API 272

DBiIsFileReferenced API 274

DBiIsServerEnabled API 276

DBiIsTableEnabled API 277

DBiPrepareAttrs API 278

DBRM (Database Request Module) 67

DBvAdminGetInaccessibleFiles API 279

DBvAdminGetReferencedFiles API 281

DBvAdminIsFileReferenced API 283

DBvDisableColumn API 285

DBvDisableServer API 287

DBvDisableTable API 288

DBvEnableColumn API 290

DBvEnableServer API 292

DBvEnableTable API 294

DBvGetError API 296

DBvGetInaccessibleFiles API 297

DBvGetReferencedFiles API 299

DBvIsColumnEnabled API 301

DBvIsFileReferenced API 303

DBvIsServerEnabled API 305

DBvIsTableEnabled API 306

DBvPlay API 307

DBvPrepareAttrs API 309

definition side deck 68

DELETE QBIC CATALOG command 115, 358

deleting data from a table 27

diagnostic information 381

directionality 17

DISABLE COLUMN command 359

DISABLE SERVER command 360

DISABLE TABLE command 361

display a video frame 103

displaying a thumbnail 105

displaying an image 103

distinct type 12

DMBSETUP, editing and running 125

DSNALI 69

Duration UDF 176

E

embedded SQL, preparing 67

ENABLE COLUMN command 362

ENABLE SERVER 363

ENABLE TABLE command 365

enabling database servers 42

environment variables 103

DB2AUDIOEXPORT 409

DB2AUDIOPATH 409

DB2AUDIOPLAYER 104

DB2AUDIOSTORE 409

DB2AUDIOTEMP 409

DB2IMAGEBROWSER 103

DB2IMAGEEXPORT 409

DB2IMAGEPATH 409

DB2IMAGESTORE 409

DB2IMAGETEMP 409

DB2VIDEOEXPORT 409

DB2VIDEOPATH 409

DB2VIDEOPLAYER 104

DB2VIDEOSTORE 409

DB2VIDEOTEMP 409

external security 41

F

features, QBIC query 123

file 64

file 64 (*continued*)

finding files referenced by tables 53

name (that contains object) 177

names, relative 66

names, specifying 66

storing from client 77

transmitting an object between a table and 64

transmitting an object to or from client 65

updating from client 96

file reference variable 65

Filename UDF 177

FindInstrument UDF 178

FindTrackName UDF 179

Format UDF 180

formats of objects 71

handled by DB2 extenders 71

identifying for storage 79

identifying for update 98

retrieving video 155

using your own for storage 81

using your own for update 99

FrameRate UDF 181

G

GET EXTENDER STATUS command 367

GET INACCESSIBLE FILES command 368

GET QBIC CATALOG INFO command 113, 370

GET REFERENCED FILES command 371

GetInstruments UDF 182

GetTrackNames UDF 183

goal mode 32

GRANT command 373

H

handle 15

header files 63

Height UDF 184

hierarchical file system (HFS) 12

histogram color 16

description 16

feature name 111

I

image 3

average color 16

colors in (number of) 190

comment attribute 153

compression type 72

conversion options 72

displaying 103

file name 177

format attribute 180

formats 71

height 184

height conversion 72

histogram color 16

identifying format for storage 79

identifying format for update 98

- image 3 (*continued*)
 - import time 186
 - importer 185
 - number of colors in 190
 - pixel 16
 - positional color 17
 - query by content 107
 - retrieving 84
 - rotation 72
 - score (QBIC) 133
 - size 207
 - storing 73
 - texture 17
 - time stored 186
 - time updated 213
 - update time 213
 - updater 212
 - updating 90
 - user ID of person who stored 185
 - user ID of person who updated 212
 - width 214
 - width conversion 72

- Image Extender 4
 - DBaPrepareAttrs API 247
 - DBiAdminGetInaccessibleFiles API 248
 - DBiAdminGetReferencedFiles API 250
 - DBiAdminIsFileReferenced API 252
 - DBiBrowse API 254
 - DBiDisableColumn API 256
 - DBiDisableServer API 258
 - DBiDisableTable API 259
 - DBiEnableColumn API 261
 - DBiEnableServer API 263
 - DBiEnableTable API 265
 - DBiGetError API 267
 - DBiGetInaccessibleFiles API 268
 - DBiGetReferencedFiles API 270
 - DBiIsColumnEnabled API 272
 - DBiIsFileReferenced API 274
 - DBiIsServerEnabled API 276
 - DBiIsTableEnabled API 277
 - DBiPrepareAttrs API 278
 - DBvPrepareAttrs API 309
 - overview 4
 - UDFs 145
 - UDTs 145
- Importer UDF 185
- ImportTime UDF 186
- include files 63
 - description 63

J

- job DMBSETUP, editing and running 125

L

- large object (LOB) 11
 - description 11
 - displaying 103
 - playing 103
 - transmitting 64
- link editing 68
- LOB (large object) 11

- LOB (large object) 11 (*continued*)
 - description 11
 - displaying 103
 - locator 65
 - playing 103
 - transmitting 64
- locator 65

M

- MaxBytesPerSec UDF 187
- media files 413
- metadata tables 14
 - description 14
 - security 32
- MIDI instrument 182
- MMDB_STORAGE_TYPE_EXTERNAL 79
 - when storing 79
 - when updating 98
- MMDB_STORAGE_TYPE_INTERNAL 79
 - when storing 79
 - when updating 98

N

- Notices 421
- NumAudioTracks UDF 188
- number of bits to represent image 72
- NumChannels UDF 189
- NumColors UDF 190
- NumFrames UDF 191
- NumVideoTracks UDF 192

O

- object 11
 - alignment of 149
 - aspect ratio of 150
 - attributes, retrieving 88
 - audio channels (number of) 189
 - audio tracks (number of) 188
 - bits per sample of audio 151
 - colors in image (number of) 190
 - comment 153
 - compression format of video 155
 - data transfer rate of audio 152
 - data transfer rate of video 187
 - description 11
 - displaying 103
 - duration of audio or video 176
 - file name 177
 - format 180
 - formats 71
 - frame rate of video 181
 - frames in video (number of) 191
 - height 184
 - import time 186
 - importer 185
 - number of audio channels 189
 - number of audio tracks 188
 - number of colors in image 190
 - number of frames in video 191
 - number of video tracks 192
 - playing 103
 - playing time of audio or video 176
 - retrieving 84
 - sampling rate of audio 206

- object 11 (*continued*)
 - security 32
 - size 207
 - storing 73
 - throughput of audio 152
 - throughput of video 181, 187
 - thumbnail 208
 - time stored 186
 - time updated 213
 - transmitting 64
 - update time 213
 - updater 212
 - updating 90
 - user ID of person who stored 185
 - user ID of person who updated 212
 - video tracks (number of) 192
 - width 214
- object orientation 11
- ODBC dynamic link library 68
- ODBC environment 67
- OPEN QBIC CATALOG command 110, 375
- operating environments for DB2
 - extenders 9
- overloaded function names 14
- overview of DB2 extenders 3
- overwrite indicator 87

P

- package, binding 67
- performance objectives 32
- photometric (image inversion) 72
- pixel 16
- planning considerations 31
- platforms for DB2 extenders 9
- playing a video 103
- playing an audio 103
- playing time of audio or video 176
- positional color 17
 - description 17
 - feature name 111
- precompiling 67
- prelinking 68
- preparing a DB2 extender application 67

Q

- QbAddFeature API 111, 310
- QbCatalogColumn API 114, 312
- QbCloseCatalog API 115, 314
- QbColor 127
- QbColorFeatureClass 111
- QbColorHistogramFeatureClass 111
- QbCreateCatalog API 108, 315
- QbDeleteCatalog API 115, 317
- QbDrawFeatureClass 111
- QbGetCatalogInfo API 113, 319
- QbHistogramColor 128
- QBIC catalog 16
- QBIC query 122
 - adding a feature to 126
 - creating 126
 - data source 126
 - deleting 131
 - description 122
 - issuing 131

- QBIC query 122 (*continued*)
 - object 125
 - removing a feature from 131
 - retrieving information about 130
 - saving 129
 - string 123
- QbImageSource 127
- QbListFeatures 113
- QbListFeatures API 320
- QbOpenCatalog API 110, 322
- QbQueryAddFeature API 126, 324
- QbQueryCreate API 126, 326
- QbQueryDelete API 131, 327
- QbQueryGetFeatureCount API 130, 328
- QbQueryGetString API 129, 329
- QbQueryListFeatures API 130, 330
- QbQueryNameCreate API 332
- QbQueryNameDelete API 131, 334
- QbQueryNameSearch API 132, 335
- QbQueryRemoveFeature API 131, 337
- QbQuerySearch API 132, 339
- QbQuerySetFeatureData API 126, 341
- QbQuerySetFeatureWeight API 343
- QbQueryStringSearch API 132, 344
- QbReCatalogColumn API 114, 346
- QbRemoveFeature API 112, 348
- QbScoreFromName UDF 133, 193
- QbScoreFromStr UDF 133, 195
- QbScoreTBFromName UDF 133, 196
- QbScoreTBFromStr UDF 133, 198
- QbTextureFeatureClass 111
- query, QBIC 122
 - building 122
 - issuing 131
- Query by Image Content (QBIC) 16
 - catalog 16
 - QbAddFeature API 310
 - QbCatalogColumn API 312
 - QbCloseCatalog API 314
 - QbCreateCatalog API 315
 - QbDeleteCatalog API 317
 - QbGetCatalogInfo API 319
 - QbListFeatures API 320
 - QbOpenCatalog API 322
 - QbQueryAddFeature API 324
 - QbQueryCreate API 326
 - QbQueryDelete API 327
 - QbQueryGetFeatureCount API 328
 - QbQueryGetString API 329
 - QbQueryListFeatures API 330
 - QbQueryNameCreate API 332
 - QbQueryNameDelete API 334
 - QbQueryNameSearch API 335
 - QbQueryRemoveFeature API 337
 - QbQuerySearch API 339
 - QbQuerySetFeatureData API 341
 - QbQuerySetFeatureWeight API 343
 - QbQueryStringSearch API 344
 - QbReCatalogColumn API 346
 - QbRemoveFeature API 348
 - steps 107
- query string, QBIC 123
 - reusing 129
- QUIT command 376

R

- recovery 35
- reference variable, file 65

- relative file names 66
- REMOVE QBIC FEATURE
 - command 112, 377
- Replace UDF 200
- ReplaceA UDF 203
- retrieving an object 84
- return codes 381
- return codes (SQLSTATE) 382
- REVOKE command 378
- rotation of image 72
- run-time environment 4

S

- sample media files 413
- sample programs 413
- sampling rate of audio 206
- SamplingRate UDF 206
- scaling factor 72
- schema name 13
- score, image (QBIC) 133
- security 32
- segment 65
- server file 64
 - retrieving to 87
 - storing from 78
 - transmitting an object between a table and 64
 - transmitting an object to 64
 - updating from 97
- service class 31
- SET CURRENT PATH statement 13
- side deck 68
- signature, function 14
- size of object 207
- Size UDF 207
- Software Developers Kit (SDK) 4
- SQLSTATE codes 382
- storing an object 73
- string, QBIC query 123

T

- tables 45
 - disabling 49
 - enabling 45
- TERMINATE command 380
- Text Extender 4
- texture 17
 - description 17
 - feature name 111
- throughput of audio 152
- throughput of video 187
- thumbnail 82
 - displaying 105
 - storing 82
 - updating 100
- Thumbnail UDF 208
- TicksPerQNote UDF 210
- TicksPerSec UDF 211
- trace facility 404
- track names, MIDI 183
- track number, MIDI 179
- track number of MIDI instrument 178
- tracks 188
 - number of audio 188
 - number of video 192

- transmitting large objects 64
- trigger 14

U

- UDF (user-defined function) 12
 - AlignValue 149
 - AspectRatio 150
 - BitsPerSample 151
 - BytesPerSec 152
 - Comment 153
 - CompressType 155
 - Content 156
 - ContentA 160
 - current path 13
 - DB2Audio 162
 - DB2AudioA 164
 - DB2Image 167
 - DB2ImageA 170
 - DB2Video 172
 - DB2VideoA 174
 - description 12
 - Duration 176
 - Filename 177
 - FindInstrument 178
 - FindTrackName 179
 - Format 180
 - FrameRate 181
 - GetInstruments 182
 - GetTrackNames 183
 - Height 184
 - Importer 185
 - ImportTime 186
 - MaxBytesPerSec 187
 - names 13
 - NumAudioTracks 188
 - NumChannels 189
 - NumColors 190
 - NumFrames 191
 - NumVideoTracks 192
 - overloaded 14
 - QbScoreFromName 193
 - QbScoreFromStr 195
 - QbScoreTBFromName 196
 - QbScoreTBFromStr 198
 - reference 145
 - Replace 200
 - ReplaceA 203
 - SamplingRate 206
 - signature 14
 - Size 207
 - Thumbnail 208
 - TicksPerQNote 210
 - TicksPerSec 211
 - Updater 212
 - UpdateTime 213
 - Width 214
- UDT (user-defined type) 12
 - description 12
 - names 13
- Unicode support 70
- Updater UDF 212
- UpdateTime UDF 213
- updating an object 90
- user-defined function 12
 - AlignValue 149
 - AspectRatio 150
 - BitsPerSample 151

user-defined function 12 (*continued*)

- BytesPerSec 152
- Comment 153
- CompressType 155
- Content 156
- ContentA 160
- current path 13
- DB2Audio 162
- DB2AudioA 164
- DB2Image 167
- DB2ImageA 170
- DB2Video 172
- DB2VideoA 174
- description 12
- Duration 176
- Filename 177
- FindInstrument 178
- FindTrackName 179
- Format 180
- FrameRate 181
- GetInstruments 182
- GetTrackNames 183
- Height 184
- Importer 185
- ImportTime 186
- MaxBytesPerSec 187
- names 13
- NumAudioTracks 188
- NumChannels 189
- NumColors 190
- NumFrames 191
- NumVideoTracks 192
- overloaded 14
- QbScoreFromName 193
- QbScoreFromStr 195
- QbScoreTBFromName 196
- QbScoreTBFromStr 198
- reference 145
- Replace 200
- ReplaceA 203
- SamplingRate 206
- signature 14
- Size 207
- Thumbnail 208
- TicksPerQNote 210
- TicksPerSec 211
- Updater 212
- UpdateTime 213
- Width 214

user-defined type (UDT) 12

- description 12
- names 13

video 3 (*continued*)

- frame rate 181
- frames in (number of) 191
- height 184
- identifying format for storage 79
- identifying format for update 98
- import time 186
- importer 185
- number of audio channels in 189
- number of audio tracks in 188
- number of frames in 191
- number of video tracks in 192
- playing 103
- playing time 176
- retrieving 84
- size 207
- storing 73
- throughput (bytes per second) 187
- throughput (frame rate) 181
- thumbnail 208
- time stored 186
- time updated 213
- update time 213
- updater 212
- updating 90
- user ID of person who stored 185
- user ID of person who updated 212
- video tracks in (number of) 192
- width 214

Video Extender 4

- DBvAdminGetInaccessibleFiles API 279
- DBvAdminGetReferencedFiles API 281
- DBvAdminIsFileReferenced API 283
- DBvDisableColumn API 285
- DBvDisableServer API 287
- DBvDisableTable API 288
- DBvEnableColumn API 290
- DBvEnableServer API 292
- DBvEnableTable API 294
- DBvGetError API 296
- DBvGetInaccessibleFiles API 297
- DBvGetReferencedFiles API 299
- DBvIsColumnEnabled API 301
- DBvIsFileReferenced API 303
- DBvIsServerEnabled API 305
- DBvIsTableEnabled API 306
- DBvPlay API 307
- overview 4
- UDFs 145
- UDTs 145

V

video 3

- alignment of 149
- aspect ratio of 150
- audio channels in (number of) 189
- audio tracks in (number of) 188
- comment attribute 153
- compression format 155
- data transfer rate 187
- duration 176
- file name 177
- format attribute 180
- formats 71

W

- wait indicator 105
- width of object 214
- Width UDF 214
- Workload Manager (WLM) 31

Readers' Comments — We'd Like to Hear from You

DB2 Universal Database for OS/390 and z/OS
Image, Audio, and Video Extenders
Administration and Programming
Version 7

Publication No. SC26-9947-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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

Name

Address

Company or Organization

Phone No.

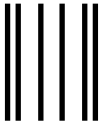


Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape



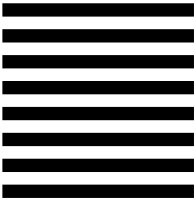
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department HHX/H3 PO Box 49023
SAN JOSE CA 95161-9023



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Program Number: 5575-DB2



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9947-00

