DB2 Universal Database for OS/390 and z/OS

Image, Audio, and Video Extenders
Administration and Programming
Image, Audio, and Video Extenders
Administration and Programming
Note
Before using this information and the product it supports, be sure to read the general information under “Notices”.

This edition applies to Version 7 of IBM DB2 Universal Database 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.

This and other books in the DB2 for OS/390 and z/OS library are periodically updated with technical changes. These updates are made available to licensees of the product on CD-ROM and on the Web (currently at www.ibm.com/software/data/db2/os390/library.html). Check these resources to ensure that you are using the most current information.

© 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. ................................................................. xi

Tables. ................................................................. xiii

About this book. ....................................................... xix
Who should use this book. ......................................... xix
How to use this book .................................................. xix
Highlighting conventions ............................................. xx
How to read the syntax diagrams .................................. xx
Accessibility ............................................................. xxi

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

Chapter 1. Overview .................................................. 3
Exploiting DB2 .......................................................... 3
Powerful new ways to search for information ................. 4
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 ........ 6
  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 ....................................................................................... 34
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
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storing an object that resides on the server</td>
<td>78</td>
</tr>
<tr>
<td>Specifying database or file storage</td>
<td>78</td>
</tr>
<tr>
<td>Identifying the format for storage</td>
<td>79</td>
</tr>
<tr>
<td>Storing an object with user-supplied attributes</td>
<td>81</td>
</tr>
<tr>
<td>Storing a thumbnail (image and video only)</td>
<td>82</td>
</tr>
<tr>
<td>Storing a comment</td>
<td>83</td>
</tr>
<tr>
<td>Retrieving an image, audio, or video object</td>
<td>84</td>
</tr>
<tr>
<td>Content UDF formats for retrieval</td>
<td>84</td>
</tr>
<tr>
<td>Retrieving an object to the client</td>
<td>85</td>
</tr>
<tr>
<td>Retrieving an object to a server file</td>
<td>86</td>
</tr>
<tr>
<td>Retrieving and using attributes</td>
<td>88</td>
</tr>
<tr>
<td>Retrieving comments</td>
<td>89</td>
</tr>
<tr>
<td>Updating an image, audio, or video object</td>
<td>90</td>
</tr>
<tr>
<td>Content UDF formats for updating</td>
<td>90</td>
</tr>
<tr>
<td>ContentA UDF formats for updating</td>
<td>92</td>
</tr>
<tr>
<td>Replace UDF formats for updating</td>
<td>93</td>
</tr>
<tr>
<td>ReplaceA UDF formats for updating</td>
<td>95</td>
</tr>
<tr>
<td>Updating an object from the client</td>
<td>96</td>
</tr>
<tr>
<td>Updating an object from the server</td>
<td>97</td>
</tr>
<tr>
<td>Specifying database or file storage for updates</td>
<td>97</td>
</tr>
<tr>
<td>Identifying the format for update</td>
<td>98</td>
</tr>
<tr>
<td>Updating an object with user-supplied attributes</td>
<td>99</td>
</tr>
<tr>
<td>Updating a thumbnail (image and video only)</td>
<td>100</td>
</tr>
<tr>
<td>Updating a comment</td>
<td>101</td>
</tr>
<tr>
<td>Chapter 11. Displaying or playing an image, audio, or video object</td>
<td>103</td>
</tr>
<tr>
<td>Using the display or play APIs</td>
<td>103</td>
</tr>
<tr>
<td>Identifying a display or play program</td>
<td>103</td>
</tr>
<tr>
<td>Specifying BLOB or file content</td>
<td>104</td>
</tr>
<tr>
<td>Specifying a wait indicator</td>
<td>105</td>
</tr>
<tr>
<td>Displaying a thumbnail-size image or video frame</td>
<td>105</td>
</tr>
<tr>
<td>Displaying a full-size image or video frame</td>
<td>106</td>
</tr>
<tr>
<td>Playing an audio or video</td>
<td>106</td>
</tr>
<tr>
<td>Chapter 12. Querying images by content</td>
<td>107</td>
</tr>
<tr>
<td>How to query by image content</td>
<td>107</td>
</tr>
<tr>
<td>Managing QBIC catalogs</td>
<td>108</td>
</tr>
<tr>
<td>Creating a QBIC catalog</td>
<td>108</td>
</tr>
<tr>
<td>Opening a QBIC catalog</td>
<td>110</td>
</tr>
<tr>
<td>Adding a feature to a QBIC catalog</td>
<td>111</td>
</tr>
<tr>
<td>Removing a feature from a QBIC catalog</td>
<td>112</td>
</tr>
<tr>
<td>Retrieving information about a QBIC catalog</td>
<td>112</td>
</tr>
<tr>
<td>Manually cataloging a column of images</td>
<td>113</td>
</tr>
<tr>
<td>Recataloging images</td>
<td>114</td>
</tr>
<tr>
<td>Closing a QBIC catalog</td>
<td>115</td>
</tr>
<tr>
<td>Deleting a QBIC catalog</td>
<td>115</td>
</tr>
<tr>
<td>QBIC catalog sample program</td>
<td>115</td>
</tr>
<tr>
<td>Building queries</td>
<td>121</td>
</tr>
<tr>
<td>Specifying a query string</td>
<td>122</td>
</tr>
<tr>
<td>Using a query object</td>
<td>124</td>
</tr>
<tr>
<td>Issuing queries by image content</td>
<td>130</td>
</tr>
<tr>
<td>Querying images</td>
<td>131</td>
</tr>
<tr>
<td>Retrieving an image score</td>
<td>132</td>
</tr>
<tr>
<td>QBIC query sample program</td>
<td>133</td>
</tr>
</tbody>
</table>
Chapter 13. User-defined types (distinct types) and user-defined functions ........................................ 147
User-defined types (distinct types) ........................................................................................................ 147
User-defined functions .......................................................................................................................... 147
AlignValue ........................................................................................................................................... 151
AspectRatio .......................................................................................................................................... 152
BitsPerSample ..................................................................................................................................... 153
BytesPerSec .......................................................................................................................................... 154
Comment .............................................................................................................................................. 155
CompressType ...................................................................................................................................... 157
Content ................................................................................................................................................... 158
ContentA .............................................................................................................................................. 162
DB2Audio ............................................................................................................................................ 164
DB2AudioA ........................................................................................................................................... 166
DB2Image ............................................................................................................................................ 169
DB2ImageA ........................................................................................................................................... 172
DB2Video ............................................................................................................................................. 174
DB2VideoA ........................................................................................................................................... 176
Duration ............................................................................................................................................... 179
Filename ............................................................................................................................................... 180
FindInstrument .................................................................................................................................... 181
FindTrackName .................................................................................................................................... 182
Format .................................................................................................................................................... 183
FrameRate ............................................................................................................................................ 184
GetInstruments ...................................................................................................................................... 185
GetTrackNames .................................................................................................................................... 186
Height ................................................................................................................................................... 187
Importer ............................................................................................................................................. 188
ImportTime ........................................................................................................................................... 189
MaxBytesPerSec .................................................................................................................................. 190
NumAudioTracks ................................................................................................................................ 191
NumChannels ....................................................................................................................................... 192
NumColors ............................................................................................................................................ 193
NumFrames .......................................................................................................................................... 194
NumVideoTracks .................................................................................................................................. 195
QbScoreFromName ............................................................................................................................... 196
QbScoreFromStr ................................................................................................................................... 198
QbScoreTBFromName ............................................................................................................................ 199
QbScoreTBFromStr ............................................................................................................................... 201
Replace .................................................................................................................................................. 203
ReplaceA ............................................................................................................................................... 206
SamplingRate ....................................................................................................................................... 209
Size ....................................................................................................................................................... 210
Thumbnail ............................................................................................................................................ 211
TicksPerQNote .................................................................................................................................... 213
TicksPerSec .......................................................................................................................................... 214
Updater .................................................................................................................................................. 215
UpdateTime ........................................................................................................................................... 216
Width ..................................................................................................................................................... 217

Chapter 14. Application programming interfaces ................................................................................. 219
DBaAdminGetInaccessibleFiles ............................................................................................................... 220
DBaAdminGetReferencedFiles ................................................................................................................ 222
DBaAdminIsFileReferenced .................................................................................................................... 224
DBaDisableColumn ............................................................................................................................... 226
QbListFeatures ............................................. 327
QbOpenCatalog .............................................. 329
QbQueryAddFeature .......................................... 331
QbQueryCreate ............................................. 333
QbQueryDelete .............................................. 334
QbQueryGetFeatureCount .................................. 335
QbQueryGetString ........................................... 336
QbQueryListFeatures ........................................ 337
QbQueryNameCreate ......................................... 339
QbQueryNameDelete ......................................... 341
QbQueryNameSearch .......................................... 342
QbQueryRemoveFeature ..................................... 344
QbQuerySearch ............................................... 346
QbQuerySetFeatureData ...................................... 348
QbQuerySetFeatureWeight .................................. 350
QbQueryStringSearch ........................................ 351
QbReCatalogColumn .......................................... 353
QbRemoveFeature ............................................ 355

Chapter 15. Administration commands for the client .................... 357
Entering DB2 Extender administration commands .......................... 357
Getting online help for DB2 Extender commands ............................ 358
ADD QBIC FEATURE ........................................ 359
CATALOG QBIC COLUMN .................................... 360
CLOSE QBIC CATALOG ....................................... 361
CREATE QBIC CATALOG ..................................... 362
DELETE QBIC CATALOG ...................................... 364
DISABLE COLUMN ........................................... 365
DISABLE SERVER ............................................ 366
DISABLE TABLE ............................................... 367
ENABLE COLUMN ............................................ 368
ENABLE SERVER ............................................. 369
ENABLE TABLE ............................................... 371
GET EXTENDER STATUS ...................................... 373
GET INACCESSIBLE FILES .................................... 374
GET QBIC CATALOG INFO .................................... 376
GET REFERENCED FILES ..................................... 377
GRANT ...................................................... 379
OPEN QBIC CATALOG ....................................... 381
QUIT ....................................................... 382
REMOVE QBIC FEATURE ..................................... 383
REVOKE ...................................................... 384
TERMINATE .................................................. 386

Chapter 16. Diagnostic information ........................................ 387
Handling UDF return codes ....................................... 387
Handling API return codes ..................................... 388
SQLSTATE codes ............................................... 388
Messages ....................................................... 392
Diagnostic tracing ............................................. 410
Start tracing ................................................... 410
Stop tracing ..................................................... 411
Reformat trace information ....................................... 411

Appendix A. Setting environment variables for DB2 Extenders .......... 413
How environment variables are used to resolve file names .......... 413
# Figures

1. A multimedia database table ......................................................... 5
2. A query that accesses videos. .......................................................... 6
3. An application that accesses and plays videos ..................................... 6
4. Searching for images by content ...................................................... 7
5. An application that searches for images by content. ............................ 8
6. Administrative support tables .......................................................... 15
7. Handles ............................................................................................ 16
8. The employee table ............................................................................ 19
9. The employee table with an audio column added ................................. 20
10. Inserting data into a table ................................................................. 24
11. Selecting data from a table ............................................................... 25
12. Displaying and playing objects .......................................................... 26
13. Updating data in a table ..................................................................... 27
14. Sample code that enables a database server ........................................ 43
15. Sample code that enables a table ....................................................... 47
16. Sample code that enables a column ................................................... 49
17. Sample code that checks if a database server is enabled ................. 52
18. Sample code that checks if a file is referenced by user tables ............... 53
19. Sample code that gets a list of referenced files .................................. 54
20. A table used in DB2 extender programming examples ............................ 61
21. An application that uses a DB2 extender ............................................ 62
22. Query by image content ................................................................. 107
23. OBIC catalog sample program ......................................................... 116
24. OBIC query sample program ............................................................ 135
25. Sample JCL ................................................................................... 419
26. Sample Bind .................................................................................. 424
27. Setting up the CLI.INI file ............................................................... 425
28. Sample STEPLIB concatenation ....................................................... 425
### Tables

1. DB2 Extender concepts ........................................... 11
2. User-defined functions created by the Image Extender ............... 21
3. User-defined functions created by the Audio Extender ............... 22
4. Character with respective extender ................................ 37
5. Administration tasks and facilities for the DB2 Extenders .......... 38
6. Tasks you can perform with DB2 extender APIs ....................... 60
7. The path name of the listfile. ..................................... 68
8. The name of the listfile. ........................................... 68
9. Formats that can be processed by the DB2 Extenders ................. 71
10. Image conversion options .......................................... 72
11. Attributes managed by the DB2 Extenders .......................... 88
12. APIs that display or play objects. .................................. 103
13. QBIC Feature Names ............................................... 111
14. Feature values that can be specified in query string ............... 122
15. The QbImageSource structure ...................................... 126
16. What the Image Extender examines in QbImageSource ............... 126
17. APIs that retrieve information about a query object ............... 129
18. APIs that query cataloged images ................................... 131
19. User-defined types created by the DB2 Extenders ................. 147
20. DB2 Extender UDFs .................................................. 148
21. The extenders that support AlignValue ............................. 151
22. The extenders that support AspectRatio ............................ 152
23. The extenders that support BitPerSample ........................... 153
24. The extenders that support BytesPerSec ............................ 154
25. The extenders that support Comment ............................... 155
26. The extenders that support CompressType ........................ 157
27. The extenders that support Content ................................ 158
28. The extenders that support ContenA ................................ 162
29. The extenders that support DB2Audio ................................ 164
30. The extenders that support DB2AudioA ............................. 166
31. The extenders that support DB2Image ................................ 169
32. The extenders that support DB2ImageA .............................. 172
33. The extenders that support DB2Video .............................. 174
34. The extenders that support DB2VideoA ............................. 176
35. The extenders that support Duration ............................... 179
36. The extenders that support Filename ............................... 180
37. The extenders that support FindInstrument ........................ 181
38. The extenders that support FindTrackName ........................ 182
39. The extenders that support Format ................................ 183
40. The extenders that support FrameRate .............................. 184
41. The extenders that support GetInstrument ......................... 185
42. The extenders that support GetTrackNames ......................... 186
43. The extenders that support Height ................................ 187
44. The extenders that support Importer ............................... 188
45. The extenders that support ImportTime ............................ 189
46. The extenders that support MaxBytesPerSec ....................... 190
47. The extenders that support NumAudioTracks ....................... 191
48. The extenders that support NumChannels .......................... 192
49. The extenders that support NumColors ............................. 193
50. The extenders that support NumFrames ............................ 194
51. The extenders that support NumVideoTracks ....................... 195
52. The extenders that support QbScoreFromName ..................... 196
53. The extenders that support QbScoreFromStr ....................... 198
54. The extenders that support QbScoreTBFromName .............................. 199
55. The extenders that support QbScoreTBFromStr .................................. 201
56. The extenders that support Replace ................................................. 203
57. The extenders that support ReplaceA ................................................. 206
58. The extenders that support SamplingRate ......................................... 209
59. The extenders that support Size ....................................................... 210
60. The extenders that support Thumbnail .............................................. 211
61. The extenders that support TicksPerQNote ....................................... 213
62. The extenders that support TicksPerSec ........................................... 214
63. The extenders that support Updater ................................................. 215
64. The extenders that support UpdateTime .......................................... 216
65. The extenders that support Width .................................................... 217
66. The extenders that support DBaAdminGetInaccessibleFiles .................. 220
67. The names of the inaccessible files ................................................ 220
68. The extenders that support DBaAdminGetReferencedFiles ................... 222
69. The names of reference files ......................................................... 222
70. The extenders that support DBaAdminIsFileReferenced ...................... 224
71. The names of reference files ......................................................... 224
72. The extenders that support DBaDisableColumn .................................. 226
73. The files with the audio data ....................................................... 226
74. The extenders that support DBaDisableServer ................................... 228
75. The files that contain audio data .................................................... 228
76. The extenders that support DBaDisableTable .................................... 229
77. The files with audio data ............................................................. 229
78. The extenders that support DBaEnableColumn ................................... 231
79. The files with audio data ............................................................. 231
80. The extenders that support DBaEnableServer ................................... 233
81. The files with audio data ............................................................. 233
82. The extenders that support DBaEnableTable ..................................... 235
83. The files with audio data ............................................................. 235
84. The extenders that support DBaGetError ......................................... 237
85. The files with audio data ............................................................. 237
86. The extenders that support DBaGetInaccessibleFiles ......................... 238
87. The files with audio data ............................................................. 238
88. The extenders that support DBaGetReferencedFiles ............................ 240
89. The audio data files ...................................................................... 240
90. The extenders that support DBaIsColumnEnabled ............................... 242
91. The audio data files ...................................................................... 242
92. The extenders that support DBaIsFileReferenced ............................... 244
93. The audio data files ...................................................................... 244
94. The extenders that support DBaIsServerEnabled ............................... 246
95. The audio data files ...................................................................... 246
96. The extenders that support DBaIsTableEnabled .................................. 247
97. The audio data files ...................................................................... 247
98. The extenders that support DBaPlay ............................................... 249
99. The audio data files ...................................................................... 249
100. The extenders that support DBaPrepareAttrs .................................... 251
101. The audio data files ...................................................................... 251
102. The extenders that support DBiAdminGetInaccessibleFiles .................. 252
103. The image files ............................................................................ 252
104. The extenders that support DBiAdminGetReferencedFiles .................. 254
105. The image files ............................................................................ 254
106. The extenders that support DBiAdminIsFileReferenced ..................... 256
107. The image files ............................................................................ 256
108. The extenders that support DBiBrowse .......................................... 258
109. The image files ............................................................................ 258
<table>
<thead>
<tr>
<th>Page Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>110.</td>
<td>The extenders that support DBiDisableColumn</td>
<td>260</td>
</tr>
<tr>
<td>111.</td>
<td>The image files</td>
<td>260</td>
</tr>
<tr>
<td>112.</td>
<td>The extenders that support DBiDisableServer</td>
<td>262</td>
</tr>
<tr>
<td>113.</td>
<td>The image files</td>
<td>262</td>
</tr>
<tr>
<td>114.</td>
<td>The extenders that support DBiDisableTable</td>
<td>263</td>
</tr>
<tr>
<td>115.</td>
<td>The image files</td>
<td>263</td>
</tr>
<tr>
<td>116.</td>
<td>The extenders that support DBiEnableColumn</td>
<td>265</td>
</tr>
<tr>
<td>117.</td>
<td>The image files</td>
<td>265</td>
</tr>
<tr>
<td>118.</td>
<td>The extenders that support DBiEnableServer</td>
<td>267</td>
</tr>
<tr>
<td>119.</td>
<td>The image files</td>
<td>267</td>
</tr>
<tr>
<td>120.</td>
<td>The extenders that support DBiEnableTable</td>
<td>269</td>
</tr>
<tr>
<td>121.</td>
<td>The image files</td>
<td>269</td>
</tr>
<tr>
<td>122.</td>
<td>The extenders that support DBiGetError</td>
<td>271</td>
</tr>
<tr>
<td>123.</td>
<td>The image files</td>
<td>271</td>
</tr>
<tr>
<td>124.</td>
<td>The extenders that support DBiGetInaccessibleFiles</td>
<td>272</td>
</tr>
<tr>
<td>125.</td>
<td>The image files</td>
<td>272</td>
</tr>
<tr>
<td>126.</td>
<td>The extenders that support DBiGetReferencedFiles</td>
<td>274</td>
</tr>
<tr>
<td>127.</td>
<td>The image files</td>
<td>274</td>
</tr>
<tr>
<td>128.</td>
<td>The extenders that support DBiIsColumnEnabled</td>
<td>276</td>
</tr>
<tr>
<td>129.</td>
<td>The image files</td>
<td>276</td>
</tr>
<tr>
<td>130.</td>
<td>The extenders that support DBiIsFileReferenced</td>
<td>278</td>
</tr>
<tr>
<td>131.</td>
<td>The image files</td>
<td>278</td>
</tr>
<tr>
<td>132.</td>
<td>The extenders that support DBiIsServerEnabled</td>
<td>280</td>
</tr>
<tr>
<td>133.</td>
<td>The image files</td>
<td>280</td>
</tr>
<tr>
<td>134.</td>
<td>The extenders that support DBiIsTableEnabled</td>
<td>281</td>
</tr>
<tr>
<td>135.</td>
<td>The image files</td>
<td>281</td>
</tr>
<tr>
<td>136.</td>
<td>The extenders that support DBiPrepareAttrs</td>
<td>283</td>
</tr>
<tr>
<td>137.</td>
<td>The image files</td>
<td>283</td>
</tr>
<tr>
<td>138.</td>
<td>The extenders that support DBvAdminGetInaccessibleFiles</td>
<td>284</td>
</tr>
<tr>
<td>139.</td>
<td>The video files</td>
<td>284</td>
</tr>
<tr>
<td>140.</td>
<td>The extenders that support DBvAdminGetReferencedFiles</td>
<td>286</td>
</tr>
<tr>
<td>141.</td>
<td>The video files</td>
<td>286</td>
</tr>
<tr>
<td>142.</td>
<td>The extenders that support DBvAdminIsFileReferenced</td>
<td>288</td>
</tr>
<tr>
<td>143.</td>
<td>The video files</td>
<td>288</td>
</tr>
<tr>
<td>144.</td>
<td>The extenders that support DBvDisableColumn</td>
<td>290</td>
</tr>
<tr>
<td>145.</td>
<td>The video files</td>
<td>290</td>
</tr>
<tr>
<td>146.</td>
<td>The extenders that support DBvDisableServer</td>
<td>292</td>
</tr>
<tr>
<td>147.</td>
<td>The video files</td>
<td>292</td>
</tr>
<tr>
<td>148.</td>
<td>The extenders that support DBvDisableTable</td>
<td>293</td>
</tr>
<tr>
<td>149.</td>
<td>The video files</td>
<td>293</td>
</tr>
<tr>
<td>150.</td>
<td>The extenders that support DBvEnableColumn</td>
<td>295</td>
</tr>
<tr>
<td>151.</td>
<td>The video files</td>
<td>295</td>
</tr>
<tr>
<td>152.</td>
<td>The extenders that support DBvEnableServer</td>
<td>297</td>
</tr>
<tr>
<td>153.</td>
<td>The video files</td>
<td>297</td>
</tr>
<tr>
<td>154.</td>
<td>The extenders that support DBvEnableTable</td>
<td>299</td>
</tr>
<tr>
<td>155.</td>
<td>The video files</td>
<td>299</td>
</tr>
<tr>
<td>156.</td>
<td>The extenders that support DBvGetError</td>
<td>301</td>
</tr>
<tr>
<td>157.</td>
<td>The video files</td>
<td>301</td>
</tr>
<tr>
<td>158.</td>
<td>The extenders that support DBvGetInaccessibleFiles</td>
<td>302</td>
</tr>
<tr>
<td>159.</td>
<td>The video files</td>
<td>302</td>
</tr>
<tr>
<td>160.</td>
<td>The extenders that support DBvGetReferencedFiles</td>
<td>304</td>
</tr>
<tr>
<td>161.</td>
<td>The video files</td>
<td>304</td>
</tr>
<tr>
<td>162.</td>
<td>The extenders that support DBvIsColumnEnabled</td>
<td>306</td>
</tr>
<tr>
<td>163.</td>
<td>The video files</td>
<td>306</td>
</tr>
<tr>
<td>164.</td>
<td>The extenders that support DBvIsFileReferenced</td>
<td>308</td>
</tr>
<tr>
<td>165.</td>
<td>The video files</td>
<td>308</td>
</tr>
</tbody>
</table>

*Tables*
| 166. | The extenders that support DBvIsServerEnabled | 310 |
| 167. | The video files | 310 |
| 168. | The extenders that support DBvIsTableEnabled | 311 |
| 169. | The video files | 311 |
| 170. | The extenders that support DBvPlay | 313 |
| 171. | The video files | 313 |
| 172. | The extenders that support DBvPrepareAttrs | 315 |
| 173. | The video files | 315 |
| 174. | The extenders that support QbAddFeature | 316 |
| 175. | The image API file | 316 |
| 176. | The extenders that support QbCatalogColumn | 318 |
| 177. | The image API file | 318 |
| 178. | The extenders that support QbCloseCatalog | 320 |
| 179. | The image API file | 320 |
| 180. | The extenders that support QbCreateCatalog | 321 |
| 181. | The image API file | 321 |
| 182. | The extenders that support QbDeleteCatalog | 323 |
| 183. | The image API file | 323 |
| 184. | The extenders that support QbGetCatalogInfo | 325 |
| 185. | The image API file | 325 |
| 186. | The extenders that support QbListFeatures | 327 |
| 187. | The image API file | 327 |
| 188. | The extenders that support QbOpenCatalog | 329 |
| 189. | The image API file | 329 |
| 190. | The extenders that support QbQueryAddFeature | 331 |
| 191. | The image query API file | 331 |
| 192. | The extenders that support QbQueryCreate | 333 |
| 193. | The image query API file | 333 |
| 194. | The extenders that support QbQueryDelete | 334 |
| 195. | The query API file | 334 |
| 196. | The extenders that support QbQueryGetFeatureCount | 335 |
| 197. | The query API file | 335 |
| 198. | The extenders that support QbQueryGetString | 336 |
| 199. | The query API file | 336 |
| 200. | The extenders that support QbQueryListFeatures | 337 |
| 201. | The query API file | 337 |
| 202. | The extenders that support QbQueryNameCreate | 339 |
| 203. | The query API file | 339 |
| 204. | The extenders that support QbQueryNameDelete | 341 |
| 205. | The query API file | 341 |
| 206. | The extenders that support QbQueryNameSearch | 342 |
| 207. | The query API file | 342 |
| 208. | The extenders that support QbQueryRemoveFeature | 344 |
| 209. | The query API file | 344 |
| 210. | The extenders that support QbQuerySearch | 346 |
| 211. | The query API file | 346 |
| 212. | The extenders that support QbQuerySetFeatureData | 348 |
| 213. | The query API file | 348 |
| 214. | The extenders that support QbQuerySetFeatureWeight | 350 |
| 215. | The query API file | 350 |
| 216. | The extenders that support QbQueryStringSearch | 351 |
| 217. | The query API file | 351 |
| 218. | The extenders that support QbReCatalogColumn | 353 |
| 219. | The image API file | 353 |
| 220. | The extenders that support QbRemoveFeature | 355 |
| 221. | The image API file | 355 |
About this book

This book describes how to use DB2 Extenders™ for z/OS™ to prepare and maintain a DB2 UDB server for z/OS 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 z/OS 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 z/OS. People who work with the Text Extender for z/OS 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 z/OS. Read this part if you are new to administering or programming with the DB2 Extenders for z/OS.

“Part 2. Administering Image, Audio, and Video Data”
This part discusses planning considerations for DB2 Extenders for z/OS. It also describes how to prepare and maintain a DB2 UDB for z/OS database server for image, audio, and video data. Read this part if you need to plan for and administer a DB2 UDB for z/OS 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 z/OS UDFs and APIs to request operations on image, audio, or video data. It also discusses considerations for building DB2 extender for z/OS 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 z/OS UDFs, APIs, administrative commands, and diagnostic information such as messages and codes. Read this part if you are familiar with DB2 extender for z/OS concepts and tasks, but need information about a specific DB2 extender for z/OS UDF, API, command, message, or code.

“Appendixes”
The appendixes describe:
– How to set environment variables that are used by the DB2 Extenders for z/OS 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.

## Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products. The major accessibility features in z/OS products enable users to:

- Use assistive technologies such as screen reader and screen magnifier software
- Operate specific or equivalent features by using only a keyboard
- Customize display attributes such as color, contrast, and font size

Assistive technology products, such as screen readers, function with the z/OS user interfaces. Consult the documentation for the assistive technology products for specific information when you use assistive technology to access these interfaces.

Online documentation is available in the DB2 Information Center, which is an accessible format when used with assistive technologies such as screen reader or screen magnifier software. The DB2 Information Center for z/OS solutions is available at the following Web site:

Part 1. Introduction

Chapter 1. Overview .................................................. 3
Exploiting DB2 ......................................................... 3
Powerful new ways to search for information .................. 4
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 ........ 6
  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 ................................. 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
Chapter 1. Overview

DB2 Universal Database™ Server for z/OS (DB2 UDB for z/OS) 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 z/OS, which are features of DB2 UDB for z/OS, help you exploit DB2 UDB for z/OS’s object-relational features. (In this book, “DB2” refers to DB2 UDB for z/OS, and either “DB2 Extenders” or simply “extenders” refers to DB2 Extenders for z/OS.) 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.

Note
The Image, Audio, and Video Extenders for OS/390 and z/OS do not support the DB2 UDB for OS/390 and z/OS data sharing function in a Parallel Sysplex environment.
New ways to search

Powerful new ways to search for information

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

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

The DB2 Extenders

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

This book covers the Image, Audio, and Video Extenders. All further references to “extenders” or “DB2 Extenders” in this book refer to the Image, Audio, and Video Extenders, unless otherwise noted. For information about the Text Extender, see Text Extender Administration and Programming. 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 z/OS 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 z/OS client using z/OS 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](image-url) contains image, audio and video data as well as traditional data types.

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 shows a query that accesses the videos. Notice that the Video Extender UDFs named Filename and Duration in the query.

```
SELECT FILENAME(ADS_VIDEO)
FROM ADS
WHERE CLIENT='IBM' AND
SHIP_DATE>='01/01/1997' AND
DURATION(ADS_VIDEO) <=30
```

*Figure 2. A query that accesses videos*

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

Figure 2 is an example of a query that the account manager can issue interactively. More typically, the account manager would use an application program to find and play videos. For example, Figure 3 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)*
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 shows searching for images by content.

![Visual example and retrieved images](image.png)

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

EXEC SQL OPEN c1;
for (;;) {
    EXEC SQL FETCH c1 INTO :hvClient, :hvCampaign, :hvSdate, :hvVid_fname;
    if (SQLCODE != 0)
        break;
    printf("Record %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)
Using the Extenders

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.

[Figure 5] shows an application that searches for images by content.

```c
#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("
    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)
```
Operating environments

The DB2 Extenders Version 8 operate with DB2 Universal Database Version 8 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 8.

The supported client platforms are: z/OS, AIX, Windows NT®, and later, Windows XP, Windows 2000, and Solaris Operating Environment.

The supported server is z/OS.
Using the Extenders
Chapter 2. DB2 extender concepts

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

Table 1. DB2 Extender concepts

<table>
<thead>
<tr>
<th>Topic</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-oriented concepts</td>
<td>Page 11</td>
</tr>
<tr>
<td>Extender data structures</td>
<td>Page 14</td>
</tr>
</tbody>
</table>

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 an associated code page. This data type is used for text objects that contain
Object-oriented concepts

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 z/OS 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.

**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
Object-oriented concepts

same signature. A signature is the qualified function name concatenated with the defined data types of all the function parameters.

Triggers

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

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

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

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

The DB2 Extenders create and maintain administrative support tables to record information about image, audio, and video data stored in a database 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 shows the administrative support tables.

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 information stored in the administrative support tables. Figure 7 on page 16 illustrates the information stored for two images in a user table.
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.
Data structures
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 contains the identification and name of each employee.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>128557</td>
<td>Anita Jones</td>
<td></td>
</tr>
<tr>
<td>843622</td>
<td>Robert Smith</td>
<td></td>
</tr>
</tbody>
</table>

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.
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 personln; /*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 DBIMAGE 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 2 on page 21](#)
### 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 the "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,,LTSPACE1
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 3.

<table>
<thead>
<tr>
<th>UDF name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignValue</td>
<td>Get the bytes per sample value of the audio</td>
</tr>
</tbody>
</table>

Table 3. User-defined functions created by the Audio Extender
Table 3. User-defined functions created by the Audio Extender (continued)

<table>
<thead>
<tr>
<th>UDF name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitsPerSample</td>
<td>Get the number of bits used to represent the audio</td>
</tr>
<tr>
<td>BytesPerSec</td>
<td>Get the average number of bytes per second of audio</td>
</tr>
<tr>
<td>Comment</td>
<td>Get or update user comments</td>
</tr>
<tr>
<td>Content</td>
<td>Get or update the content of an audio</td>
</tr>
<tr>
<td>ContentA</td>
<td>Update the content of an audio with user-supplied attributes</td>
</tr>
<tr>
<td>DB2Audio</td>
<td>Store the content of an audio</td>
</tr>
<tr>
<td>DB2AudioA</td>
<td>Store the content of an audio with user-supplied attributes</td>
</tr>
<tr>
<td>Duration</td>
<td>Get the playing time of an audio</td>
</tr>
<tr>
<td>Filename</td>
<td>Get the name of the file that contains an audio</td>
</tr>
<tr>
<td>FindInstrument</td>
<td>Get the number of the audio track that records a specific instrument in an audio</td>
</tr>
<tr>
<td>FindTrackName</td>
<td>Get the track number of a named track in an audio recording</td>
</tr>
<tr>
<td>Format</td>
<td>Get the audio format</td>
</tr>
<tr>
<td>GetInstruments</td>
<td>Get the names of the instruments recorded in an audio</td>
</tr>
<tr>
<td>GetTrackNames</td>
<td>Get the track names in an audio</td>
</tr>
<tr>
<td>Importer</td>
<td>Get the user ID of the importer of an audio</td>
</tr>
<tr>
<td>ImportTime</td>
<td>Get the timestamp when an audio was imported</td>
</tr>
<tr>
<td>NumAudioTracks</td>
<td>Get the number of recorded tracks in an audio</td>
</tr>
<tr>
<td>NumChannels</td>
<td>Get the number of audio channels</td>
</tr>
<tr>
<td>Replace</td>
<td>Update the content and user comments for an audio recording</td>
</tr>
<tr>
<td>ReplaceA</td>
<td>Update the content and user comments for an audio recording with user-supplied attributes</td>
</tr>
<tr>
<td>SamplingRate</td>
<td>Get the sampling rate of the audio</td>
</tr>
<tr>
<td>Size</td>
<td>Get the size of an audio in bytes</td>
</tr>
<tr>
<td>TicksPerQNote</td>
<td>Get the number of clock ticks per quarter note used in recording an audio</td>
</tr>
<tr>
<td>TicksPerSec</td>
<td>Get the number of clock ticks per second used in recording an audio</td>
</tr>
<tr>
<td>Updater</td>
<td>Get the user ID of the updater of an audio</td>
</tr>
<tr>
<td>UpdateTime</td>
<td>Get the timestamp when an audio was updated</td>
</tr>
</tbody>
</table>

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 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.
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.

```sql
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(  "employee/images/ajones.bmp" /*image Extender UDF*/
        CURRENT SERVER, /*database server name in*/
        CURRENT SERVER register*/
        /"employee/images/ajones.bmp" /*image source file*/
        hvInt_Stor, /*store image in DB as BLOB*/
        /*comment*/
        DB2AUDIO(  "employee/sounds/ajones.wav", /*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*/
            /*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 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.

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

EXEC SQL SELECT IMPORTTIME(PICTURE), IMPORTTIME(VOICE) 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.

```sql
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.

```sql
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:

```sql
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 .................................................... 34
    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 can improve performance. However there is extra overhead in maintaining the multiple address spaces associated with two WLM environments.
Workload management considerations

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.
Security considerations

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 z/OS 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, z/OS Open Edition Services calls an external security product such as Security Server (RACF) to get the user ID (UID) 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.
Security considerations

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 the file system” 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 219 for the authority required by DB2 extender administrative APIs. See Chapter 15, “Administration commands for the client,” on page 357 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 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.
Backup and recovery considerations

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 219 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 357 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 5 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:

<table>
<thead>
<tr>
<th>Character with respective extender</th>
<th>Extender</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Audio</td>
</tr>
<tr>
<td>i</td>
<td>Image</td>
</tr>
<tr>
<td>v</td>
<td>Video</td>
</tr>
</tbody>
</table>

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 5. Administration tasks and facilities for the DB2 Extenders

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

**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 DB2xxxx 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 147.

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, gbpcache-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(dmbs,"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]="MMDBSYSG"; /* define global meta tablespace */
    char wlm[8+1]="WLMENV1"; /* define wlm environment */
    char security[8+1]="DB2"; /* define external security */

    ...
Enabling database servers

```c
/*--- 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)
Using the db2ext command line:

```c
****** 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
", 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.
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.

displays sample code that enables a table.
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 tsp1[8+1]="SAMPLI"; /* define lob tablespace for image */
char tspblk[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, tsp1i, (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 --------------------------*/
Enabling tables

/*---- enable table for image extender --------------------------*/
printf("%s: Enabling table......\n", program);
step="DBiEnableTable";
if (SERVER_IS_DB2390)
{
    sprintf((char*) tspblk, "%s, %s, %s, %s",
        tsp, (char*)stmt04, tspli, (char*) stmt04);
    rc = DBiEnableTable(tspblk, tableName);
    if (rc < 0) {
        printf("%s: %s failed!\n", program, step);
        printMsg(rc);
        DB_iGetError(&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);
        DB_iGetError(&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,,ltbspace1

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 CREATEB 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.
Using the db2ext command line:

In this example, the column already exists, and the database server and table are enabled.

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 357. For more information about the extender APIs, see Chapter 14, “Application programming interfaces,” on page 219.

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.
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.
Listing referenced files

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 DBxAdminGetReferencedFiles 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.

```c
/*---- Query the database server using DBiAdminIsFileReferenced API. ----*/
step=DBiAdminIsFileReferenced((char*) uid, filename, &count, &filelist);
if (rc < 0) {
    printf("%s: %s FAILED\n", program, step);
    printMsg(rc);
    DBiGetError(&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", argv[0], step);
}
```

Figure 18. Sample code that checks if a file is referenced by user tables
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 ("\htable = %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", 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 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.
Chapter 11. Displaying or playing an image, audio, or video object

Chapter 12. Querying images by content
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 6 lists the tasks that you can perform with the extender UDFs and APIs and shows where each task is described.

Table 6. Tasks you can perform with DB2 extender APIs

<table>
<thead>
<tr>
<th>Task</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store an image, audio, or video object</td>
<td>Page 73</td>
</tr>
<tr>
<td>Retrieve an image, audio, or video object</td>
<td>Page 84</td>
</tr>
<tr>
<td>Retrieve and use image, audio, and video attributes</td>
<td>Page 88</td>
</tr>
<tr>
<td>Retrieve comments associated with an image, audio, or video object</td>
<td>Page 89</td>
</tr>
<tr>
<td>Update an image, audio, or video object</td>
<td>Page 90</td>
</tr>
<tr>
<td>Display an image object</td>
<td>Page 103</td>
</tr>
<tr>
<td>Display a thumbnail-size image or video frame</td>
<td>Page 105</td>
</tr>
<tr>
<td>Play an audio or video object</td>
<td>Page 106</td>
</tr>
<tr>
<td>Query images by content</td>
<td>Page 107</td>
</tr>
</tbody>
</table>

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
);

![Employee table](image)

**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_hd1` 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.
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.

Figure 21 shows an application that uses a DB2 extender.

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

int count=0;

long main(int argc,char *argv[])
{
  EXEC SQL BEGIN DECLARE SECTION;
    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;

  // Application logic
}
```

Figure 21. An application that uses a DB2 extender (Part 1 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:

```c
#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:

```c
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,
      ...)
```
Before you begin

```
hv:audio,
'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 a z/OS 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).
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 z/OS 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 413 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 z/OS UNIX file.
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 a z/OS 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 an 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 a z/OS 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

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 387](#).

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 219](#) for details.
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 z/OS, the DLLs or shared libraries are specified in export files.

When you prelink and link edit on a z/OS 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 a z/OS 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 a z/OS 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:

Table 7. The path name of the listfile.

<table>
<thead>
<tr>
<th>Path</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/lpp/db2ext/lib</td>
<td>AIX</td>
</tr>
<tr>
<td>install_path/bin (the default install path is c:\dmb)</td>
<td>Windows NT, 98, 95</td>
</tr>
<tr>
<td>/opt/IBMdb2ex/V6.1/lib</td>
<td>Solaris Operating Environment</td>
</tr>
</tbody>
</table>

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

Table 8. The name of the listfile.

<table>
<thead>
<tr>
<th>Listfile</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmbmvsb1.lst</td>
<td>AIX</td>
</tr>
<tr>
<td>dmbmvsb3.lst</td>
<td>Windows NT, 98, 95</td>
</tr>
<tr>
<td>dmbmvsb8.lst</td>
<td>Solaris Operating Environment</td>
</tr>
</tbody>
</table>

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 z/OS 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:

dsnal("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:

dsnal("OPEN", "V61A", "MYPLAN", rc, reascode)

Configuring the ODBC Initialization file
If your application runs in a DB2 ODBC environment on z/OS, 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:

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

Chapter 9. Programming overview 69
Before you begin

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 9 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 9. Formats that can be processed by the DB2 Extenders

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM</td>
<td>PS/2® Audio Video Connection (AVC)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>BMP</td>
<td>OS/2 - Microsoft Windows bitmap¹</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EPS</td>
<td>Encapsulated PostScript</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>EP2</td>
<td>Encapsulated level 2 PostScript</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>GIF</td>
<td>Compuserve GIF89a (including animated GIFs²) and 87</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IMG</td>
<td>IOCA image</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IPS</td>
<td>Brooktrout FAX card file</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JPG</td>
<td>JPEG³ (JFIF format)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PCX</td>
<td>PC paint file (grayscale only)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PGM</td>
<td>Portable gray map (from PBMPLUS)</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PS</td>
<td>PostScript</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PSC</td>
<td>Compressed PostScript image</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PS2</td>
<td>PostScript level 2 (color)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TIF</td>
<td>All TIFF 5.0 formats</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>YUV</td>
<td>Digital video for YUV</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Audio formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIF or AIFF</td>
<td>Audio Interchange File Format</td>
<td>X</td>
</tr>
<tr>
<td>AIFFC</td>
<td>Audio Interchange File Format Compressed</td>
<td>X</td>
</tr>
<tr>
<td>AU</td>
<td>Sun audio file format</td>
<td>X</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1998, 2001
Table 9. Formats that can be processed by the DB2 Extenders (continued)

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDI</td>
<td>Musical Instrument Digital Interface</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>MPG1 or MPEG1</td>
<td>Moving Pictures Expert Group 1</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>WAV or WAVE</td>
<td>Wave</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

**Video formats**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVI</td>
<td>Audio/Video Interleaved</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>MPG1 or MPEG1</td>
<td>Motion Picture Coding Expert Group 1</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>MPG2 or MPEG2</td>
<td>Motion Picture Coding Expert Group 2</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>QT</td>
<td>Quicktime (AVI)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Image conversion options

Table 10 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 10. Image conversion options

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-b</td>
<td>Number of bits used to represent each image sample</td>
<td>1 or 8 bits</td>
</tr>
<tr>
<td>-s^4</td>
<td>Scaling factor</td>
<td>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.</td>
</tr>
</tbody>
</table>
| -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 |
| -r^4      | Rotation                                          | 0 = 0 degrees (no rotation)             |
|           |                                                   | 1 = 90 degrees (counterclockwise)       |
|           |                                                   | 2 = 90 degrees (clockwise)              |
|           |                                                   | 3 = 180 degrees                         |

1. Read is supported for Windows Version 2, Windows Version 3, and Windows NT 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 10. Image conversion options (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-x&lt;sup&gt;4&lt;/sup&gt;</td>
<td>Width in pixels</td>
<td>Number of pixels</td>
</tr>
<tr>
<td>-y&lt;sup&gt;4&lt;/sup&gt;</td>
<td>Height in pixels</td>
<td>Number of pixels</td>
</tr>
<tr>
<td>-c</td>
<td>Compression type</td>
<td>0 = IBM MMR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = CCITT Group 3 1-D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = CCITT Group 3 2-D (k=2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = CCITT Group 3 2-D (k=4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = CCITT Group 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 = TIFF Type 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = Uncompressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 = LZW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 = TIFF Packbits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25 = JBIG</td>
</tr>
</tbody>
</table>

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.

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.

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

**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:

```c
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:

```c
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:

```c
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:

```c
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(  
    CURRENT_SERVER,  /*Image Extender UDF*/  
    '/employee/images/ajones.bmp',  /*source file */  
    'ASIS',  /*keep the image format*/  
    hvStorageType  /*store image in DB as BLOB*/  
  )  /*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(  
    CURRENT_SERVER,  /*Image Extender UDF*/  
    '/employee/images/ajones.bmp',  /*source file */  
    'ASIS',  /*keep the 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 xxxx 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) */
);                    /* or NULL for storage in file server */
```

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) */
);                    /* or NULL for storage in file server */
```

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 hvImattrs[100];
EXEC SQL END DECLARE SECTION;

DB2IMAGEATTRS  *pimgattr;

hvStorageType=MMDB_STORAGE_TYPE_INTERNAL;
pimgattr = (DB2IMAGEATTRS *) hvImattrs;
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 */
'Format', /* source format */
',') /* no thumbnail */
);

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 a z/OS 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.

```c
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 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.

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

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 413.

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 9 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
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',
        */
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 9 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 9 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 10 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 z/OS 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.

```c
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 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.

```c
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, /* user-specified attributes */
'', /* no tracknames */
'', /* no instrument names */
'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',
    DB2VIDEO(
        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',
    '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 */);
Retrieving 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 */
    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 a z/OS 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, 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.
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 a z/OS 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 9 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 10 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.

```c
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 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 9 on page 7 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, /* 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.
Using attributes

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 11 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 11. Attributes managed by the DB2 Extenders. You can access each attribute through its UDF.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>UDF</th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name of server file in which the object is stored</td>
<td>Filename</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>User ID of person who stored the object</td>
<td>Importer</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Date and time when the object was stored</td>
<td>ImportTime</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Size of the object in bytes</td>
<td>Size</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>User ID of person who last updated the object</td>
<td>Updater</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Date and time when the object was last updated</td>
<td>UpdateTime</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Format of the object (for example, GIF or MPEG1)</td>
<td>Format</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Comments about the object</td>
<td>Comment</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Height of the object (in pixels)</td>
<td>Height</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Width of the object (in pixels)</td>
<td>Width</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of colors in the object</td>
<td>NumColors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thumbnail-size image of the object</td>
<td>Thumbnail</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of bytes returned per sample in an audio, or in an audio track of a video</td>
<td>AlignValue</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of bits used to represent each sample</td>
<td>BitsPerSample</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of recorded channels</td>
<td>NumChannels</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Duration (in seconds)</td>
<td>Duration</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sampling rate (in samples per second)</td>
<td>SamplingRate</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average bytes per second transfer time</td>
<td>BytesPerSec</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of audio track for instrument</td>
<td>FindInstrument</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Track number of named track</td>
<td>FindTrackName</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Name of recorded instruments</td>
<td>GetInstruments</td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Table 11. Attributes managed by the DB2 Extenders (continued). You can access each attribute through its UDF.

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

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:

```sql
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:

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

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

Retrieving comments

Use the Comment UDF to retrieve comments 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.

```sql
EXEC SQL BEGIN DECLARE SECTION;
    char hvComment[16385];
    short indComment;
```
Retrieving comments

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 '%touched 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 9 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:

```java
Content(
  handle,        /* object handle */
  content,       /* object content */
```
Updating

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:
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, /* stortype */
  :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 */
  stortype, /* MMDB_STORAGE_TYPE_EXTERNAL=store */
  /* in file server */
  /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB */
  comment, /* user comment */
);

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

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 */
);
Updating

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 */
            /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB*/
    comment     /* 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 */
            /* MMDB_STORAGE_TYPE_INTERNAL=store as a BLOB*/
    comment     /* user comment */
);

For example, the following statements in a C application program update an audio clip in the employee table and update its associated comment. The source content for the update is in a server file 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 */
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,
'/employee/newimg/ajones.bmp',
'AJONES\'s new picture')
WHERE NAME='Anita Jones';

**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 (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 (image and video only) */
);
```

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.
Updating

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,
      '/employee/newvid/ajones.mpg',
      :hvStorageType,
      'Anita''s new video')
      :hvVidattrs,
   'FormatV',
   'MPEG1',
      :hvThumbnail)
   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 a z/OS 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
      'WAVE',
      ')')
   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
    '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,
    '/employee/newimg/ajones.bmp',
    '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 413.

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.

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

EXEC SQL UPDATE EMPLOYEE
  SET PICTURE=REPLACE(
Updating

PICTURE,
:img_buf,
'ASIS',
'/employee/newimg/ajones.bmp', /*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)
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*/
  'BMP')
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 9 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 9 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 10 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',  /*source format*/
       'GIF',  /*target format*/
       'r',    /*conversion options*/
   )
WHERE NAME='Anita Jones';

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

EXEC SQL UPDATE EMPLOYEE
   SET PICTURE=CONTENT(
       PICTURE,
       '/employee/newimg/ajones.bmp',
       'BMP',  /*source format*/
       'GIF',  /*target format*/
       'r 90', /*conversion options*/
   )
WHERE NAME='Anita Jones';

Updating an object with user-supplied attributes

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

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.

```sql
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,
  '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.

```sql
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(
      '/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';

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;
Updating

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 a z/OS client, you need to be a z/OS 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:

Table 12. APIs that display or play objects.

<table>
<thead>
<tr>
<th>Use this API</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBIBrowse</td>
<td>Display an image or video frame</td>
</tr>
<tr>
<td>DBaPlay</td>
<td>Play an audio clip</td>
</tr>
<tr>
<td>DBvPlay</td>
<td>Play a video clip or display a video frame</td>
</tr>
</tbody>
</table>

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.

You can also specify a null value instead of naming a specific display or play program. In this case, the extender starts the default image browser, audio player, or video player named in the DB2IMAGEBROWSER, DB2AUDIOPLAYER, or DB2VIDEOPLAYER environment variables. For more information about how the DB2 Extenders use environment variables, see [Appendix A, “Setting environment variables for DB2 Extenders,” on page 413].

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

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,
    "/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 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 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
Displaying thumbnails

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:

```sql
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.

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 images to be searched. In either case, you can submit the query from the DB2 command line or from within a program.

5. The Image Extender includes software that is developed by the University of California, Berkeley, and its contributors.
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 113 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 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.

```sql
SQLINTEGER autoCatalog=0;
rc=QbCreateCatalog(
```
Managing QBIC catalogs

"employee",
"picture",
autoCatalog,
"qbtbspace");

**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:
Managing QBIC catalogs

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 13).

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QbColorFeatureClass</td>
<td>Average color</td>
</tr>
<tr>
<td>QbColorHistogramFeatureClass</td>
<td>Histogram color</td>
</tr>
<tr>
<td>QbDrawFeatureClass</td>
<td>Postional color</td>
</tr>
<tr>
<td>QbTextureFeatureClass</td>
<td>Texture</td>
</tr>
</tbody>
</table>

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 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;
```
Managing QBIC catalogs

```c
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:

```c
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.

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:

```c
typedef struct{
    char    tableName[qbiMaxTableName+1]
    char    columnName[qbiMaxColumnName+1]
    SQLINTEGER featureCount;
    SQLINTEGER autoCatalog;
} 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 qbiMaxFeature name constant is used to calculate the buffer size for the QbListFeatures API:

```c
long bufSize;
long count;
char *featureNames;
QbCatalogHandle CatHdl;
QbCatalogInfo catInfo;
/* Get user table name, image column name, feature count, */
rc=QbGetCatalogInfo(
    CatHdl,
    &catInfo);
/* List feature names */
bufSize=catInfo.featureCount*qbiMaxFeatureName;
featureNames=malloc(bufSize);
rc=QbListFeatures(
    CatHdl,
    bufSize
    count,
    featureNames);
```

**Using the command line:** The GET QBIC CATALOG INFO command acts on the currently open catalog. In the following example, the command is used to retrieve information about the currently open QBIC catalog:

```
GET QBIC CATALOG INFO
```

**Manually cataloging 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
Managing QBIC catalogs

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:

```c
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:

```c
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 113).

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:

```c
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:

```c
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 116 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 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 417.

Note the following points in the program:

1. Include the dmbqapi header file.
2. Connect to the database.
Managing QBIC catalogs

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 <sqlcl.h>
#include <sqlcli1.h>
#include <dmbqbqpi.h>
#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;
SQLENV henv;
SQLDBC 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)
    }
```
/----- 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);  
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)
Managing QBIC catalogs

```c
/**************************************************************/
/** createCatalog()  ****
/**  ****
/**************************************************************/
SQLINTEGER createCatalog()
{
    SQLINTEGER rc = MMDB_SUCCESS;
    SQLINTEGER autoCatalog = 0;
    char * tabespaces = 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");

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

    rc = SQLGetInfo( hdbc, SQL_DBMS_NAME, SQLPOINTER) &dbms_name, sizeof(dbms_name), &dmbs_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
                        tabespaces = "SAMPQBIC"; /* set 1 tablespace for qbic tables */
                        SQLFreeStmt(hstmt, SQL_DROP);
                }
}

Figure 23. QBIC catalog sample program (Part 3 of 6)
/ *-----------------------------*/
/ * now create the qbic catalog */
/ *-----------------------------*/
if ( rc == MMDB_SUCCESS )
{
  rc = QbCreateCatalog( S
    (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);
}

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

```c
/*************************************************
****  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 ... 
");
  if(chdl) /* if we have an open catalog, else do nothing */
  {
    printf("  Color Feature Class ... 
");
    rc = QbAddFeature(
          chdl,
          CfeatureName
    ) ;
    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);
  }
}
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 ...
");
  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

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.
Building queries

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 123. 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 QbScoreTBFFromStr) 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 130 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 quotations 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.QbScoreTBFFromStr ('texture file=<server,patterns/ptrn07.gif>', 'fabric', 'swatch_img', 10)) as T1"
```

Table 14 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>
<thead>
<tr>
<th>Feature name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>averageColor, average, or</td>
<td>color=&lt;Rvalue, Gvalue, Bvalue&gt;</td>
</tr>
<tr>
<td>QbColorFeatureClass</td>
<td>Each color value is an integer from 0 to 255 that identifies the</td>
</tr>
<tr>
<td></td>
<td>red value (Rvalue), green value (Gvalue), and blue value (Bvalue) of</td>
</tr>
<tr>
<td></td>
<td>the image.</td>
</tr>
<tr>
<td></td>
<td>file=&lt;file_location, filename&gt;</td>
</tr>
<tr>
<td></td>
<td>The file_location is server for a server file. The filename is</td>
</tr>
<tr>
<td></td>
<td>file name specified in the format appropriate for the system in which</td>
</tr>
<tr>
<td></td>
<td>the file resides, or a relative file name. DB2 Extenders resolves</td>
</tr>
<tr>
<td></td>
<td>the relative file name using environment variables (see How</td>
</tr>
<tr>
<td></td>
<td>environment variables are used to resolve file names on page 413).</td>
</tr>
</tbody>
</table>
Building queries

Table 14. Feature values that can be specified in query string (continued)

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

**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:

```
image_handle=<(image_file_location, image_filename), ...>
```
Building queries

```
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 using a query string" on page 128). 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
QbScoreTBFFromName) 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 130 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 SYSEFLT 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
Building queries

the 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:

```c
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:

```c
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 126 for details).
Building queries

**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:

```c
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>qbiSource_ImageHandle</td>
<td>Source is in a user table column</td>
</tr>
<tr>
<td>qbiSource_AverageColor</td>
<td>Source is an average color specification</td>
</tr>
<tr>
<td>qbiSource_HistogramColor</td>
<td>Source is a histogram color specification</td>
</tr>
</tbody>
</table>

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 16.

**Table 16. What the Image Extender examines in QbImageSource**

<table>
<thead>
<tr>
<th>Source</th>
<th>What the Image Extender examines</th>
<th>Where specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>a user table</td>
<td>image handle</td>
<td>image handle field of QbImageSource</td>
</tr>
<tr>
<td>average color specification</td>
<td>red, green, and blue color values</td>
<td>QbColor structure (see page 126 for details about using this structure)</td>
</tr>
<tr>
<td>histogram color specification</td>
<td>color values and percentages</td>
<td>QbHistogramColor structure (see page 127 for details about using this structure)</td>
</tr>
</tbody>
</table>

**QbColor:** Use the QbColor structure to specify the red, green, and blue values of an average color when the data source is an average color specification. The structure is defined in the include (header) file for QBIC, dmbqbapi.h, as follows:
typedef struct{
    SQLUSMALLINT red; /*0 off - 65535 (fully on) */
    SQLUSMALLINT green; /*0 off - 65535 (fully on) */
    SQLUSMALLINT blue; /*0 off - 65535 (fully on) */
} QbColor;

Set the values in QbColor to indicate the amount of red, green, and blue pixels to be factored in the average value calculation. The values can range from 0 to 65535. A value of 0 means ignore the entry.

**QbHistogramColor:** Use the QbHistogramColor structure to specify each color component of a histogram color specification. The full specification for a histogram color is contained in an array of QbHistogramColor structures. Each structure contains a color value and a percentage. The color value is comprised of red, green, and blue pixel values. The percentage specifies the percentage of that color that is required in the target image.

The structure is defined in the include (header) file for QBIC, dmbqbapi.h, as follows:

```c
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.

```c
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:

```c
char featureName[qbiMaxFeatureName];
QbColor avgColor;
QbImageSource imgSource;

imgSource.type=qbiSource_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 */
```
Building queries

**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:

```c
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 QbScoreTBFromStr and QbScoreTBFromUDFs UDFs in other queries by image content (see "Issuing queries by image content" on page 130).

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

- QbQueryCreate
- QbQueryAddFeature
- QbQuerySetFeatureData
- QbQuerySetFeatureWeight
• QbQueryRemoveFeature

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:

```c
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.

<table>
<thead>
<tr>
<th>Use this API</th>
<th>To retrieve</th>
</tr>
</thead>
<tbody>
<tr>
<td>QbQueryGetFeatureCount</td>
<td>The number of features in a query object</td>
</tr>
<tr>
<td>QbQueryListFeatures</td>
<td>The names of the features in a query object</td>
</tr>
</tbody>
</table>

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:

```c
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:

```c
SQLINTEGER retCount,bufSize;
char* featureName;
QbQueryHandle qHandle;

bufSize=qbiMaxFeatureName;
```
Building queries

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 122) or in a query object (see "Using a query object" on page 124).

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.
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:

<table>
<thead>
<tr>
<th>API</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>QbQueryStringSearch</td>
<td>Query string</td>
</tr>
<tr>
<td>QbQuerySearch</td>
<td>Query object handle</td>
</tr>
<tr>
<td>QbQueryNameSearch</td>
<td>Query object name</td>
</tr>
</tbody>
</table>

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:

```c
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 the cataloged images in a table column. Notice that the pointer to the query scope is set to a zero value.

```c
QbResult     returns[MaxQueryReturns];
SQLINTEGER   maxResults=qbiMaxQueryReturns;
SQLINTEGER   count;
QbQueryHandle qHandle;
QbResult     results[qbiMaxQueryReturns];

rc=QbQueryStringSearch(
    "QbColorFeatureClass color=<255, 0, 0>" /*query string */
    "employee",
    "picture",
    maxResults,
    0,
    qbiArray,
    &count,
    results); /*array of returned results*/
```
Issuing QBIC queries

Here is a request that uses the QbQuerySearch API. Notice that the query object handle is specified as input.

```c
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.
Issuing QBIC queries

SELECT name, description
decimal (QbScoreFromStr(swatch_img,
   'QbColorFeatureClass color=<255, 0, 0>', /* query string */
   10, 5) AS score
FROM fabric /* table column */
ORDER BY score

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 135 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 417.
Issuing QBIC queries

Note the following points in Figure 24 on page 135:

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

/*
1234567890123456789012345678901234567890123456789012345678901234567890 */
"
"#---------------------------------------------------------------*
"! 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

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

```c
}  
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;
    }
    arg1 = nextWord(0);
    if (arg1 == 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);  
_QbQuerySearch(averageHandle, tableName, columnName, MaxQueryReturns,  
    0, 0, &imageCount, results);  
lastHandle = averageHandle;

    currentImage = -1;
}
```

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

Image, Audio, and Video Extenders Administration and Programming
void commandLoop(void)
{
    int done = 0;
    while (!done) {
        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)
Issuing QBIC queries

```c
int main(int argc, char *argv[]) {
    char *inst;
    int i;
    #if defined(DMB_HP)
    _main();
    #endif
    if ( ( argc > 4 ) || ( argc >=2 ) && ( strcmp(argv[1],"?")== 0 ) )
    {
        printf("Syntax for qbicdemo \\
                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");
        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, (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)
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);

printf("Adding 'Draw' feature to Query object . . .\n");
_QbQueryAddFeature(drawHandle, "QbDrawFeatureClass");

printf("Starting query commands . . .\n");
commandLoop();

printf("Deleting query objects . . .\n");
_QbQueryDelete(drawHandle);  
_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)
Issuing QBIC queries
### Chapter 13. User-defined types (distinct types) and user-defined functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>147</td>
</tr>
<tr>
<td>User-defined types (distinct types)</td>
<td>147</td>
</tr>
<tr>
<td>User-defined functions.</td>
<td>147</td>
</tr>
<tr>
<td>AlignValue</td>
<td>151</td>
</tr>
<tr>
<td>AspectRatio</td>
<td>152</td>
</tr>
<tr>
<td>BitsPerSample</td>
<td>153</td>
</tr>
<tr>
<td>BytesPerSec</td>
<td>154</td>
</tr>
<tr>
<td>Comment</td>
<td>155</td>
</tr>
<tr>
<td>CompressType</td>
<td>157</td>
</tr>
<tr>
<td>Content</td>
<td>158</td>
</tr>
<tr>
<td>ContentA</td>
<td>162</td>
</tr>
<tr>
<td>DB2Audio</td>
<td>164</td>
</tr>
<tr>
<td>DB2AudioA</td>
<td>166</td>
</tr>
<tr>
<td>DB2Image</td>
<td>169</td>
</tr>
<tr>
<td>DB2ImageA</td>
<td>172</td>
</tr>
<tr>
<td>DB2Video</td>
<td>174</td>
</tr>
<tr>
<td>DB2VideoA</td>
<td>176</td>
</tr>
<tr>
<td>Duration</td>
<td>179</td>
</tr>
<tr>
<td>Filename</td>
<td>180</td>
</tr>
<tr>
<td>FindInstrument</td>
<td>181</td>
</tr>
<tr>
<td>FindTrackName</td>
<td>182</td>
</tr>
<tr>
<td>Format</td>
<td>183</td>
</tr>
<tr>
<td>FrameRate</td>
<td>184</td>
</tr>
<tr>
<td>GetInstruments</td>
<td>185</td>
</tr>
<tr>
<td>GetTrackNames</td>
<td>186</td>
</tr>
<tr>
<td>Height</td>
<td>187</td>
</tr>
<tr>
<td>Importer</td>
<td>188</td>
</tr>
<tr>
<td>ImportTime</td>
<td>189</td>
</tr>
<tr>
<td>MaxBytesPerSec</td>
<td>190</td>
</tr>
<tr>
<td>NumAudioTracks</td>
<td>191</td>
</tr>
<tr>
<td>NumChannels</td>
<td>192</td>
</tr>
<tr>
<td>NumColors</td>
<td>193</td>
</tr>
<tr>
<td>NumFrames</td>
<td>194</td>
</tr>
<tr>
<td>NumVideoTracks</td>
<td>195</td>
</tr>
<tr>
<td>QbScoreFromName</td>
<td>196</td>
</tr>
<tr>
<td>QbScoreFromStr.</td>
<td>198</td>
</tr>
<tr>
<td>QbScoreTBFFromName</td>
<td>199</td>
</tr>
<tr>
<td>QbScoreTBFFromStr.</td>
<td>201</td>
</tr>
<tr>
<td>Replace</td>
<td>203</td>
</tr>
<tr>
<td>ReplaceA</td>
<td>206</td>
</tr>
<tr>
<td>SamplingRate</td>
<td>209</td>
</tr>
<tr>
<td>Size</td>
<td>210</td>
</tr>
<tr>
<td>Thumbnail</td>
<td>211</td>
</tr>
<tr>
<td>TicksPerQNote</td>
<td>213</td>
</tr>
<tr>
<td>TicksPerSec</td>
<td>214</td>
</tr>
<tr>
<td>Updater</td>
<td>215</td>
</tr>
<tr>
<td>UpdateTime</td>
<td>216</td>
</tr>
<tr>
<td>Width</td>
<td>217</td>
</tr>
</tbody>
</table>

### Chapter 14. Application programming interfaces

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBaAdminGetInaccessibleFiles</td>
<td>219</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1998, 2001
QbCreateCatalog .................................................. 321
QbDeleteCatalog .................................................. 323
QbGetCatalogInfo .................................................. 325
QbListFeatures .................................................... 327
QbOpenCatalog .................................................... 329
QbQueryAddFeature ............................................. 331
QbQueryCreate .................................................... 333
QbQueryDelete ..................................................... 334
QbQueryGetFeatureCount ....................................... 335
QbQueryGetString ............................................... 336
QbQueryListFeatures ............................................. 337
QbQueryNameCreate ............................................. 339
QbQueryNameDelete ............................................. 341
QbQueryNameSearch ............................................. 342
QbQueryRemoveFeature ......................................... 344
QbQuerySearch .................................................... 346
QbQuerySetFeatureData ......................................... 348
QbQuerySetFeatureWeight ....................................... 350
QbQueryStringSearch ........................................... 351
QbReCatalogColumn .............................................. 353
QbRemoveFeature .................................................. 355

Chapter 15. Administration commands for the client ............... 357
Entering DB2 Extender administration commands .................. 357
Getting online help for DB2 Extender commands .................. 358
ADD QBIC FEATURE ................................................. 359
CATALOG QBIC COLUMN .......................................... 360
CLOSE QBIC CATALOG ............................................ 361
CREATE QBIC CATALOG .......................................... 362
DELETE QBIC CATALOG .......................................... 364
DISABLE COLUMN .................................................. 365
DISABLE SERVER .................................................. 366
DISABLE TABLE .................................................... 367
ENABLE COLUMN ................................................... 368
ENABLE SERVER .................................................... 369
ENABLE TABLE ...................................................... 371
GET EXTENDER STATUS .......................................... 373
GET INACCESSIBLE FILES ....................................... 374
GET QBIC CATALOG INFO ......................................... 376
GET REFERENCED FILES ......................................... 377
GRANT ............................................................. 379
OPEN QBIC CATALOG ............................................. 381
QUIT ................................................................. 382
REMOVE QBIC FEATURE ......................................... 383
REVOKE ............................................................ 384
TERMINATE .......................................................... 386

Chapter 16. Diagnostic information .................................. 387
Handling UDF return codes ....................................... 387
Handling API return codes ....................................... 388
SQLSTATE codes .................................................... 388
Messages ............................................................ 392
Diagnostic tracing .................................................. 410
Start tracing ........................................................ 410
Stop tracing .......................................................... 411
Reformat trace information ....................................... 411

Part 4. Reference 145
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 19 lists and describes the user-defined types created by the DB2 Extenders. It also lists the DB2 source data type for each UDT.

Table 19. User-defined types created by the DB2 Extenders

<table>
<thead>
<tr>
<th>UDT</th>
<th>Source data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2IMAGE</td>
<td>VARCHAR(250)</td>
<td>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.</td>
</tr>
<tr>
<td>DB2AUDIO</td>
<td>VARCHAR(250)</td>
<td>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.</td>
</tr>
<tr>
<td>DB2VIDEO</td>
<td>VARCHAR(250)</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

Table 20 on page 148 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.
## User-defined functions

### Table 20. DB2 Extender UDFs

<table>
<thead>
<tr>
<th>UDF</th>
<th>Description</th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
<th>See page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignValue</td>
<td>Returns the number of bytes per sample in a WAVE audio, or in an audio track of a video.</td>
<td></td>
<td>x</td>
<td>x</td>
<td>151</td>
</tr>
<tr>
<td>AspectRatio</td>
<td>Returns the aspect ratio of the first track of an MPEG1 and MPEG2 video.</td>
<td></td>
<td></td>
<td>x</td>
<td>152</td>
</tr>
<tr>
<td>BitsPerSample</td>
<td>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.</td>
<td>x</td>
<td>x</td>
<td></td>
<td>153</td>
</tr>
<tr>
<td>BytesPerSec</td>
<td>Returns the data transfer rate, in average bytes per second, for a WAVE audio.</td>
<td></td>
<td>x</td>
<td></td>
<td>154</td>
</tr>
<tr>
<td>Comment</td>
<td>Returns or updates a comment stored with an image, audio, or video.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>155</td>
</tr>
<tr>
<td>CompressType</td>
<td>Returns the compression format, such as MPEG-1, of a video.</td>
<td></td>
<td></td>
<td>x</td>
<td>157</td>
</tr>
<tr>
<td>Content</td>
<td>Retrieves or updates the content of an image, audio, or video from a database.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>158</td>
</tr>
<tr>
<td>ContentA</td>
<td>updates the content of an image, audio, or video with user-supplied attributes from a database.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>162</td>
</tr>
<tr>
<td>DB2Audio</td>
<td>Stores the content of an audio in a database table.</td>
<td></td>
<td>x</td>
<td></td>
<td>164</td>
</tr>
<tr>
<td>DB2AudioA</td>
<td>Stores the content of an audio with user-supplied attributes in a database table.</td>
<td></td>
<td>x</td>
<td></td>
<td>166</td>
</tr>
<tr>
<td>DB2Image</td>
<td>Stores the content of an image in a database table.</td>
<td></td>
<td>x</td>
<td></td>
<td>169</td>
</tr>
<tr>
<td>DB2ImageA</td>
<td>Stores the content of an image with user-supplied attributes in a database table.</td>
<td></td>
<td>x</td>
<td></td>
<td>172</td>
</tr>
<tr>
<td>DB2Video</td>
<td>Stores the content of a video in a database table.</td>
<td></td>
<td>x</td>
<td></td>
<td>174</td>
</tr>
<tr>
<td>DB2VideoA</td>
<td>Stores the content of a video with user-supplied attributes in a database table.</td>
<td></td>
<td>x</td>
<td></td>
<td>176</td>
</tr>
<tr>
<td>Duration</td>
<td>Returns the duration (that is, playing time in seconds) of a WAVE or AIFF audio, or video.</td>
<td>x</td>
<td>x</td>
<td></td>
<td>179</td>
</tr>
<tr>
<td>Filename</td>
<td>Returns the name of the server file that contains the contents of an image, audio, or video.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>180</td>
</tr>
<tr>
<td>FindInstrument</td>
<td>Returns the track number of the first occurrence of a specified instrument in a MIDI audio.</td>
<td></td>
<td></td>
<td>x</td>
<td>181</td>
</tr>
<tr>
<td>FindTrackName</td>
<td>Returns the number of a specified named track in a MIDI audio.</td>
<td></td>
<td></td>
<td>x</td>
<td>182</td>
</tr>
</tbody>
</table>
### User-defined functions

<table>
<thead>
<tr>
<th>UDF</th>
<th>Description</th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
<th>See page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>Returns the format of an image, audio, or video.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>FrameRate</td>
<td>Returns the throughput of a video in frames per second.</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>GetInstruments</td>
<td>Returns the instrument name of all instruments in a MIDI audio.</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GetTrackNames</td>
<td>Returns the name of all tracks in a MIDI audio.</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Height</td>
<td>Returns the height, in pixels, of an image or video frame.</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Importer</td>
<td>Returns the user ID of the person who stored an image, audio, or video in a database table.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>ImportTime</td>
<td>Returns a timestamp that indicates when an image, audio, or video was stored in a database table.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>MaxBytesPerSec</td>
<td>Returns the maximum throughput of a video in bytes per second.</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NumAudioTracks</td>
<td>Returns the number of audio tracks in a video or MIDI audio.</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NumChannels</td>
<td>Returns the number of recorded audio channels in a WAVE or AIFF audio, or video.</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NumColors</td>
<td>Returns the number of colors in an image.</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NumFrames</td>
<td>Returns the number of frames in a video.</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NumVideoTracks</td>
<td>Returns the number of video tracks in a video.</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>QbScoreFromName</td>
<td>Returns the score of an image (uses a named query object). (Replaces QbScore.)</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QbScoreFromStr</td>
<td>Returns the score of an image (uses a query string).</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QbScoreTBFromName</td>
<td>Returns a table of scores from an image column (uses a named query object).</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>QbScoreTBFromStr</td>
<td>Returns a table of scores from an image column (uses a query string).</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replace</td>
<td>Updates the content of an image, audio, or video stored in a database, and updates its comment.</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>ReplaceA</td>
<td>Updates the content of an image, audio, or video with user-supplied attributes stored in a database, and updates its comment.</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
### User-defined functions

<table>
<thead>
<tr>
<th>UDF</th>
<th>Description</th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
<th>See page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SamplingRate</td>
<td>Returns the sampling rate of a WAVE or AIFF audio, or of an audio track in a video, in number of samples per second.</td>
<td></td>
<td>x</td>
<td>x</td>
<td>209</td>
</tr>
<tr>
<td>Size</td>
<td>Returns the size of an image, audio, or video, in bytes.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>210</td>
</tr>
<tr>
<td>Thumbnail</td>
<td>Returns or updates a thumbnail-size version of an image or video frame stored in a database.</td>
<td>x</td>
<td></td>
<td>x</td>
<td>211</td>
</tr>
<tr>
<td>TicksPerQNote</td>
<td>Returns the clock speed of a recorded MIDI audio, in ticks per quarter note.</td>
<td></td>
<td>x</td>
<td></td>
<td>213</td>
</tr>
<tr>
<td>TicksPerSec</td>
<td>Returns the clock speed of a recorded MIDI audio, in ticks per second.</td>
<td></td>
<td>x</td>
<td></td>
<td>214</td>
</tr>
<tr>
<td>Updater</td>
<td>Returns the user ID of the person who last updated an image, audio, or video in a database table.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>215</td>
</tr>
<tr>
<td>UpdateTime</td>
<td>Returns a timestamp that indicates when an image, audio, or video in a database table was last updated.</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>216</td>
</tr>
<tr>
<td>Width</td>
<td>Returns the width in pixels of an image or video frame.</td>
<td>x</td>
<td></td>
<td>x</td>
<td>217</td>
</tr>
</tbody>
</table>
AlignValue

Table 21. The extenders that support AlignValue

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

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

\[ \text{AlignValue}(-\text{handle}) \]

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

 Table 22. The extenders that support AspectRatio

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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**

*Table 23. The extenders that support BitPerSample*

<table>
<thead>
<tr>
<th></th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 24. The extenders that support BytesPerSec

<table>
<thead>
<tr>
<th></th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Audio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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;
```
Table 25. The extenders that support Comment

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Returns or updates a comment that is stored with an image, audio, or video.

**Include file**

<table>
<thead>
<tr>
<th>image</th>
<th>dmbimage.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio</td>
<td>dmbaudio.h</td>
</tr>
<tr>
<td>video</td>
<td>dmbvideo.h</td>
</tr>
</tbody>
</table>

**Syntax**

**Retrieve comment**

```
GRAB-Comment---(---handle---)
```

**Syntax**

**Update comment**

```
GRAB-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;
```
/* Get the old comment */
EXEC SQL SELECT COMMENT(VIDEO) INTO :hvRemarks
    FROM EMPLOYEE
    WHERE NAME = 'Anita Jones';

/* Update the comment */
strcopy (hvRemarks, "Updated video");

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

Table 26. The extenders that support CompressType

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the compression format, such as MPEG-1, of a video.

Include file
dmbvideo.h

Syntax

\[
\text{CompressType}(\text{handle})
\]

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:

\[
\begin{align*}
\text{EXEC SQL BEGIN DECLARE SECTION;} & \\
& \quad \text{char hvVid_fname[255];} \\
\text{EXEC SQL END DECLARE SECTION;} & \\
\text{EXEC SQL SELECT FILENAME(VIDEO)} & \\
& \quad \text{INTO :hvVid_fname} \\
& \quad \text{FROM EMPLOYEE} \\
& \quad \text{WHERE COMPRESSIONTYPE(VIDEO) = 'MPEG1';}
\end{align*}
\]
Table 27. The extenders that support Content

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>dmbimage.h</td>
<td></td>
</tr>
<tr>
<td>audio</td>
<td>dmbaudio.h</td>
<td></td>
</tr>
<tr>
<td>video</td>
<td>dmbvideo.h</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

**Retrieve content to buffer or workstation client file**

```c
->Content(handle)
```

**Syntax**

**Retrieve a segment of content to buffer or workstation client file**

```c
->Content(handle, offset, size)
```

**Syntax**

**Retrieve content to server file**

```c
->Content(handle, target_file, overwrite)
```

**Syntax**

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

```c
->Content(handle, target_format)
```

**Syntax**

**Retrieve content to server file with format conversion—image only**

```c
->Content(handle, target_file, overwrite, target_format)
```

**Syntax**

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

```c
->Content(handle, target_format, conversion_options)
```
Syntax
Retrieve content to server file with format conversion and additional changes—image only

```plaintext
Content (handle-, target_file-, overwrite-, target_format-, conversion_options-)
```

Syntax
Update content from buffer or workstation client file

```plaintext
Content (handle-, content-, source_format-, target_file-)
```

Syntax
Update content from server file

```plaintext
Content (handle-, source_file-, source_format-, stortype-)
```

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

```plaintext
Content (handle-, content-, source_format-, target_file-)
```

Syntax
Update content from server file with format conversion—image only

```plaintext
Content (handle-, source_file-, source_format-, target_file-)
```

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

```plaintext
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

```plaintext
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 10 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';
Table 28. The extenders that support ContentA

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

```c
ContentA(handle, content
```n
```
target_file, attrs, format, compress_type, thumbnail)
```

Syntax

Update content with user-supplied attributes from server file

```c
ContentA(handle, source_file, stortype, attrs
```n
```
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';
Table 29. The extenders that support DB2Audio

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

```c
//DB2Audio(dbname, content, format, target_file, comment)
```

Syntax
Store content from server file

```c
//DB2Audio(dbname, source_file, format, storype, 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 9 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'));
The extender that supports DB2AudioA 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

```sql
DB2AudioA(dbname, content, target_file, comment, attrs, tracknames, instruments, format);
```

**Syntax**
Store content with user-supplied attributes from server file

```sql
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 9 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).

commontype (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 *paudioattr;
    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,
        )
    )

Chapter 13. User-defined types (distinct types) and user-defined functions 167
Table 31. The extenders that support DB2Image

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```c
DB2Image(dbname, content, source_format, target_file, comment)
```

Syntax
Store content from server file

```c
DB2Image(dbname, source_file, source_format, stortype, comment)
```

Syntax
Store content from buffer or workstation client file with format conversion

```c
DB2Image(dbname, content, source_format, target_format, target_file, comment)
```

Syntax
Store content from server file with format conversion

```c
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

```c
DB2Image(dbname, content, source_format)
```
DB2Image

Syntax
Store content from server file with format conversion and additional changes

\[-DB2Image\left(-dbname-, -source_file-, -source_format-, -target_file-, -target_format-\right)\]

Parameters (data type)
dbname (VARCHAR(18))
The name of the currently connected database, as indicated by the CURRENT SERVER special register.

ccontent (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 9 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 DB2IMAGESSTORE 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 10 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'));
Table 32. The extenders that support DB2ImageA

<table>
<thead>
<tr>
<th></th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

ccontent (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 9 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:

```sql
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',
        ''
    )
);
```
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

```sql
DB2Video(dbname, content, format, target_file, comment)
```

Syntax
Store content from server file

```sql
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.

ccontent (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 9 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:

```sql
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:

```sql
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'));
```
Table 34. The extenders that support DB2VideoA

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

```c
```

Syntax
Store content with user-supplied attributes from server file

```c
```

Parameters (data type)
dbnam (VARCHAR(18))
The name of the currently connected database, as indicated by the CURRENT SERVER special register.

ccontent (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 9 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).
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:

```sql
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,
```
DB2VideoA

'FormatV',
'MPEG1',
:hvThumbnail})
};
Duration

Table 35. The extenders that support Duration

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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;
```
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

```sql
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:

```sql
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

Table 37. The extenders that support FindInstrument

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Audio</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Video</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns the track number of the first occurrence of a specified instrument in a MIDI audio.

Include file
dmbaudio.h

Syntax

```sql
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';
```

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:

```sql
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

Table 38. The extenders that support FindTrackName

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Returns the number of a specified named track in a MIDI audio.

Include file
dmbaudio.h

Syntax

\[ \text{FindTrackName}((\text{handle}, \text{trackname})) \]

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:

```sql
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

Table 39. The extenders that support Format

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the format of an image, audio, or video.

Include file

<table>
<thead>
<tr>
<th>Include File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>dmbimage.h</td>
</tr>
<tr>
<td>audio</td>
<td>dmbaudio.h</td>
</tr>
<tr>
<td>video</td>
<td>dmbvideo.h</td>
</tr>
</tbody>
</table>

Syntax

```sql
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 9 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:

```sql
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

**FrameRate**

*Table 40. The extenders that support FrameRate*

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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:

```sql
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

Table 41. The extenders that support GetInstruments

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns instrument name of all instruments in a MIDI audio.

Include file
dmbaudio.h

Syntax

\[\text{GetInstruments} \left( \text{handle} \right)\]

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:

\[
\text{EXEC SQL BEGIN DECLARE SECTION;}
\text{char hvAud_Instr[1536];}
\text{EXEC SQL END DECLARE SECTION;}
\text{EXEC SQL SELECT GETINSTRUMENTS(SOUND)}
\text{INTO :hvAud_Instr}
\text{FROM EMPLOYEE}
\text{WHERE NAME = 'Robert Smith';}
\]
GetTrackNames

Table 42. The extenders that support GetTrackNames

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the name of all tracks in a MIDI audio.

Include file
dmbaudio.h

Syntax

```sql
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:

```sql
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

Table 43. The extenders that support Height

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

```sql
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;
```
Table 44. The extenders that support Importer

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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

Table 45. The extenders that support ImportTime

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

```sql
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:

```sql
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

MaxBytesPerSec

Table 46. The extenders that support MaxBytesPerSec

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 47. The extenders that support NumAudioTracks

<table>
<thead>
<tr>
<th></th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Video</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the number of audio tracks in a video or MIDI audio.

Include file

audio dmbaudio.h
video dmbvideo.h

Syntax

```sql
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:

```sql
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

Table 48. The extenders that support NumChannels

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the number of recorded audio channels in a WAVE or AIFF audio, or video.

Include file

audio  dmbaudio.h
video  dmbvideo.h

Syntax

```sql
--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):

```sql
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**

*Table 49. The extenders that support NumColors*

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

NumFrames

Table 50. The extenders that support NumFrames

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the number of frames in a video.

Include file
dmbvideo.h

Syntax

```sql
 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:

```sql
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

Table 51. The extenders that support NumVideoTracks

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the number of video tracks in a video.

Include file
dmbvideo.h

Syntax

```sql
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:

```sql
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

QbScoreFromName

Table 52. The extenders that support QbScoreFromName

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Depreciated 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:
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;

QbScoreFromName
QbScoreFromStr

Table 53. The extenders that support QbScoreFromStr

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Audio</td>
<td>Video</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

\[\text{QbScoreFromStr}(\text{imgHandle}, \text{query})\]

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 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:

\[\text{SELECT name FROM fabric WHERE (QbScoreFromStr(Swatch_Img, 'QbColorFeatureClass color=<255, 0, 0>')<0.1)\]
QbScoreTBFromName

Table 54. The extenders that support QbScoreTBFromName

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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—)
```

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

Table 55. The extenders that support QbScoreTBFromStr

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```sql
QbScoreTBFromStr(query, table, column)
```

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

```sql
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));
Table 56. The extenders that support Replace

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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—,)
```
Replace

```
\>
-\text{-target\_format-,-target\_file-,-conversion\_options-,-comment-})\>
```

**Syntax**

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

```
\>
-\text{-Replace-}(-\text{-handle-,-source\_file-,-source\_format-},\text{-\ldots})\>
-\text{-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.

*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

Table 57. The extenders that support ReplaceA

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

```c
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

```c
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 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',
   

WHERE NAME='Anita Jones';
SamplingRate

Table 58. The extenders that support SamplingRate

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

\[\text{SamplingRate} \left( \text{\_handle\_} \right)\]

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

**Table 59. The extenders that support Size**

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extenders</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the size of an image, audio, or video, in bytes.

**Include file**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>dmbimage.h</td>
</tr>
<tr>
<td>audio</td>
<td>dmbaudio.h</td>
</tr>
<tr>
<td>video</td>
<td>dmbvideo.h</td>
</tr>
</tbody>
</table>

**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

Table 60. The extenders that support Thumbnail

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>updates</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thumbnail-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>version</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>an</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>image</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>video</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>that</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stored</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>database.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Include file**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>dmbimage.h</td>
<td></td>
</tr>
<tr>
<td>video</td>
<td>dmbvideo.h</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

### Retrieve a thumbnail

```
Thumbnail(handle)
```

### 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:

```sql
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:

```sql
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';
Table 61. The extenders that support TicksPerQNote

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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

**Include file**
dmbaudio.h

**Syntax**

```sql
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:

```sql
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

Table 62. The extenders that support TicksPerSec

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Audio</td>
<td>Video</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 TICKS_PER_SEC(SOUND)<50;
```
Updater

Table 63. The extenders that support Updater

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the user ID of the person who last updated an image, audio, or video in a database table.

Include file

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>dmbimage.h</td>
</tr>
<tr>
<td>audio</td>
<td>dmbaudio.h</td>
</tr>
<tr>
<td>video</td>
<td>dmbvideo.h</td>
</tr>
</tbody>
</table>

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

UpdateTime

Table 64. The extenders that support UpdateTime

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Returns a timestamp that indicates when an image, audio, or video in a database table was last updated.

Include file

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>dmbimage.h</td>
</tr>
<tr>
<td>audio</td>
<td>dmbaudio.h</td>
</tr>
<tr>
<td>video</td>
<td>dmbvideo.h</td>
</tr>
</tbody>
</table>

Syntax

```sql
UPDATE_TIME(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:

```sql
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

Table 65. The extenders that support Width

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns the width in pixels of an image or video frame.

Include file

image dmbimage.h
video dmbvideo.h

Syntax

\[
\text{Width} \left( \text{handle} \right)
\]

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;
Width
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

Table 66. The extenders that support DBaAdminGetInaccessibleFiles

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 67. The names of the inaccessible files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

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:

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

rc = DBaAdminGetInaccessibleFiles("rsmith",
    &count, &filelist);
```
DBaAdminGetReferencedFiles

Table 68. The extenders that support DBaAdminGetReferencedFiles

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 69. The names of reference files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
<td></td>
</tr>
</tbody>
</table>

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:

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

rc = DBaAdminGetReferencedFiles("ajones",
    &count, &fileList);
```
DBaAdminIsFileReferenced

Table 70. The extenders that support DBaAdminIsFileReferenced

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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

Table 71. The names of reference files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

Include file

dmbaudio.h

Syntax

```c
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:

```c
#include <dmbaudio.h>
long idx;
rc = DBaAdminIsFileReferenced(NULL,
    "/audios/asmith.wav",
    &count, &tableList);
```
DBaDisableColumn

Table 72. The extenders that support DBaDisableColumn

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 73. The files with the audio data

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

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):

```c
#include <dmbaudio.h>

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

Table 74. The extenders that support DBaDisableServer

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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

Table 75. The files that contain audio data

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

Include file
dmbaudio.h

Syntax

```c
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):

```c
#include <dmbaudio.h>

int rc = DBaDisableServer();
```
DBaDisableTable

Table 76. The extenders that support DBaDisableTable

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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 CREATE privilege on the target database or is the DBADM for the database

Library file

Table 77. The files with audio data

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaugh.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaugh.so</td>
</tr>
</tbody>
</table>

Include file
dmbaudio.h

Syntax

```c
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.
**DBaDisableTable**

**MMDB_RC_NOT_CONNECTED**
Application does not have valid connection to a database server.

**Examples**

Disable the employee table for audio (DB2Audio data):

```c
#include <dmbaugh.h>

rc = DBaDisableTable("employee");
```
DBaEnableColumn

Table 78. The extenders that support DBaEnableColumn

<table>
<thead>
<tr>
<th></th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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 CREATE 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

Table 79. The files with audio data

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmdbauido.lib</td>
<td>libdmbaudio.so</td>
<td></td>
</tr>
</tbody>
</table>

Include file
dmbaudio.h

Syntax

```c
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.
DBaEnableColumn

**MMDB_RC_NO_AUTH**
Caller does not have the proper access authority.

**MMDB_WARN_ALREADY_ENABLED**
Column is already enabled.

**MMDB_RC_WRONG_SIGNATURE**
Data type for the specified column is incorrect. User-defined data type MMDBSYS.DB2AUDIO is expected.

**MMDB_RC_COLUMN_DOESNOT_EXIST**
Column is not defined in the specified table.

**MMDB_RC_NOT_CONNECTED**
Application does not have valid connection to a database 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");
```
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**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbaudio.h

**Syntax**

```c
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
DBaEnableServer

that WLM environment. If two WLM environments are specified, the second is used to run UDFs that store, retrieve, or update objects (such as 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 z/OS 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_BC_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:

```c
#include <dmbaudio.h>

rc = DBaEnableServer("myts","dmbwlm1",NULL);
```
DBaEnableTable

Table 82. The extenders that support DBaEnableTable

<table>
<thead>
<tr>
<th></th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 83. The files with audio data

<table>
<thead>
<tr>
<th></th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

Include file

dmbaudio.h

Syntax

```c
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:

```c
#include <dmbaudio.h>

rc = DBaEnableTable("myts,,mylobts",
     "employee");
```
**DBaGetError**

*Table 84. The extenders that support DBaGetError*

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns a description of the last error. Call this API after any other API returns an error code.

**Authorization**

None.

**Library file**

*Table 85. The files with audio data*

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbaudio.h

**Syntax**

```c
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`:

```c
#include <dmbaudio.h>

rc = DBaGetError(&errCode, &errMsg);
```
**DBaGetInaccessibleFiles**

Table 86. The extenders that support DBaGetInaccessibleFiles

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

Table 87. The files with audio data

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbaudio.h

**Syntax**

```c
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:

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

rc = DBaGetInaccessibleFiles("employee",
                           &count, &filelist);
```
DBaGetReferencedFiles

Table 88. The extenders that support DBaGetReferencedFiles

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 89. The audio data files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

Include file

dmbaudio.h

Syntax

```c
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);
**DBaIsColumnEnabled**

**Table 90. The extenders that support DBaIsColumnEnabled**

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

**Table 91. The audio data files**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIx</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbaudio.h

**Syntax**

```c
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:

```c
#include <dmbaudio.h>

rc = DBaIsColumnEnabled("employee",
    "sound", &status);
```
DBaIsFileReferenced

Table 92. The extenders that support DBaIsFileReferenced

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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

Table 93. The audio data files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

Include file

dmbaudio.h

Syntax

```c
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:
```c
#include <dmbaudio.h>
long idx;
rc = DBaIsFileReferenced(NULL,
   "/audios/ajones.wav",
   &count, &tableList);
```
**DBaIsServerEnabled**

**Table 94. The extenders that support DBaIsServerEnabled**

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

**Table 95. The audio data files**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbaudio.h

**Syntax**

```c
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:

```c
#include <dmbaudio.h>

rc = DBaIsServerEnabled(&status);
```
DBaIsTableEnabled

Table 96. The extenders that support DBaIsTableEnabled

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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

Table 97. The audio data files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

Include file

dmbaudio.h

Syntax

```c
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MMDB_IS_ENABLED</td>
</tr>
<tr>
<td>0</td>
<td>MMDB_IS_NOT_ENABLED</td>
</tr>
<tr>
<td>-1</td>
<td>MMDB_INVALID_DATATYPE</td>
</tr>
</tbody>
</table>

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.
DBaIsTableEnabled

Examples

Determine if the employee table is enabled for audio (DB2Audio data):
#include <dmbaudio.h>
rc = DBaIsTableEnabled("employee", &status);
DBaPlay

Table 98. The extenders that support DBaPlay

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

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

Table 99. The audio data files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

Include file

dmbaudio.h

Syntax

Play an audio stored in a column

```c
long DBaPlay(
    char *playerName,
    MMDB_PLAY_HANDLE, 
    DB2Audio *audioHandle,  
    waitFlag
);
```

Syntax

Play an audio stored as a file
DBaPlay

```c
long DBaPlay(
    char *playerName,
    MMDB_PLAY_HANDLE,  // A constant that indicates the audio is stored as a BLOB.
    char *fileName,
    waitFlag
);
```

Parameters

**playerName (in)**

The name of the audio player. If set to NULL, the default audio player specified by the DB2AUDIOPLAYER environment variable is used.

**MMDB_PLAY_HANDLE (in)**

A constant that indicates the audio is stored as a BLOB.

**MMDB_PLAY_FILE (in)**

A constant that indicates the audio is stored as a file that is accessible from the client.

**audioHandle (in)**

The handle of the audio. This parameter must be passed when you play an audio clip in a column. If the audio handle represents an external file, the client environment variable DB2VIDEOPATH is used to resolve the file location.

**fileName (in)**

The name of the file that contains the audio.

**waitFlag (in)**

A constant that indicates whether your program waits for the user to close the player before continuing. MMDB_PLAY_WAIT runs the player in the same thread as your application. MMDB_PLAY_NO_WAIT runs the player in a separate thread.

Error codes

**MMDB_SUCCESS**

API call processed successfully.

**MMDB_RC_NO_AUTH**

Caller does not have the proper access authority.

**MMDB_RC_NOT_CONNECTED**

Application does not have valid connection to a database server.

Examples

Play the audio that is identified by the audioHandle. Run the default player in the same thread as the application:

```c
#include <dmbaudio.h>

rc = DBaPlay(NULL, MMDB_PLAY_HANDLE,
              audioHandle, MMDB_PLAY_WAIT);
```
DBaPrepareAttrs

Table 100. The extenders that support DBaPrepareAttrs

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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 z/OS 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

Table 101. The audio data files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBAUDIO</td>
<td>libdmbaudio.a</td>
<td>dmbaudio.lib</td>
<td>libdmbaudio.so</td>
</tr>
</tbody>
</table>

Include file
dmbaudio.h

Syntax

```c
void DBaPrepareAttrs(
    MMDBAudioAttrs *audAttr
);```

Parameters

audAttr (in)
The user-supplied attributes of the audio.

Examples

Prepare user-supplied audio attributes:

```c
#include <dmbaudio.h>

DBaPrepareAttrs(&imgattr);
```
DBAdminGetInaccessibleFiles

Table 102. The extenders that support DBAdminGetInaccessibleFiles

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 103. The image files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file
dmbimage.h

Syntax

```c
long DBAdminGetInaccessibleFiles(
    char *qualifier,
    long *count,
    FILERT (*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.
**Examples**

List all inaccessible files that are referred to in image columns of tables that are owned by user ID rjones:

```c
#include <dmbimage.h>
long idx;

rc = DBiAdminGetInaccessibleFiles
   ("rjones", &count, &filelist);
```
DBiAdminGetReferencedFiles

Table 104. The extenders that support DBiAdminGetReferencedFiles

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 105. The image files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file
dmbimage.h

Syntax

```c
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:

```c
#include <dmbimage.h>
long idx;
rc = DBiAdminGetReferencedFiles("ajones",
    &count, &filelist);
```
The extenders that support DBiAdminIsFileReferenced

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

Table 107. The image files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbimage.h

**Syntax**

```c
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.
**Examples**

List the entries in image columns in all tables in the current database that refer to file /images/asmith.bmp:

```c
#include <dmbimage.h>
long idx;

rc = DBiAdminIsFileReferenced(NULL,
       "/images/asmith.bmp",
       &count, &tableList);
```
DBiBrowse

Table 108. The extenders that support DBiBrowse

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 109. The image files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file

dmbimage.h

Syntax

Browse an image stored in a column

```c
long DBiBrowse(
    char *browserName,
    MMDB_PLAY_HANDLE,
    DB2Image *imageHandle,
    waitFlag
);
```

Browse an image stored as a file

```c
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:

```c
#include <dmbimage.h>

rc = DBiBrowse(NULL, MMDB_PLAY_HANDLE, imageHandle, MMDB_PLAY_WAIT);
```
DBiDisableColumn

Table 110. The extenders that support DBiDisableColumn

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 111. The image files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file
dmbimage.h

Syntax

```c
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):
```c
#include <dmbimage.h> rc = DBiDisableColumn("employee", "picture");
```
DBiDisableServer

Table 112. The extenders that support DBiDisableServer

<table>
<thead>
<tr>
<th>Extender</th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 113. The image files

<table>
<thead>
<tr>
<th>OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>DMBIMAGE</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
<tr>
<td>AIX</td>
<td>libdmbimage.a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>dmbimage.lib</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solaris</td>
<td>libdmbimage.so</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Include file
dmbimage.h

Syntax

```c
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):

```c
#include <dmbimage.h>

int rc = DBiDisableServer();
```
**DBiDisableTable**

*Table 114. The extenders that support DBiDisableTable*

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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**

*Table 115. The image files*

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbimage.h

**Syntax**

```c
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):

```c
#include <dmbimage.h>

rc = DBiDisableTable("employee");
```
DBiEnableColumn

Table 116. The extenders that support DBiEnableColumn

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 117. The image files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file
dmbimage.h

Syntax

```c
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.
DBiEnableColumn

**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**
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:

```c
#include <dmbimage.h>

rc = DBiEnableColumn("employee", "picture");
```
DBiEnableServer

Table 118. The extenders that support DBiEnableServer

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 119. The image files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file
dmbimage.h

Syntax

```c
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
DBiEnableServer

is used to run UDFs that import or export objects (such as 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 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 z/OS 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","dmbwl1",NULL);
```
DBiEnableTable

Table 120. The extenders that support DBiEnableTable

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 121. The image files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file

dmbimage.h

Syntax

```c
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
DBiEnableTable

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, 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

*Table 122. The extenders that support DBiGetError*

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns a description of the last error. Call this API after any other API returns an error code.

#### Authorization

None.

#### Library file

*Table 123. The image files*

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

#### Include file

dmbimage.h

#### Syntax

```c
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`:

```c
#include <dmbimage.h>

rc = DBiGetError(&errCode, &errMsg);
```
DBiGetInaccessibleFiles

**Table 124. The extenders that support DBiGetInaccessibleFiles**

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

**Table 125. The image files**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbimage.h

**Syntax**

```c
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:

```c
#include <dmbimage.h>
long idx;

rc = DBiGetInaccessibleFiles("employee",
       &count, &filelist);
```
DBiGetReferencedFiles

Table 126. The extenders that support DBiGetReferencedFiles

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 127. The image files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file

dmbimage.h

Syntax

```c
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:

```c
#include <dmbimage.h>
long idx;

rc = DBiGetReferencedFiles("employee",
    &count, &filelist);
```
DBIsColumnEnabled

Table 128. The extenders that support DBIsColumnEnabled

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 129. The image files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

Include file

dmbimage.h

Syntax

```c
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MMDB_IS_ENABLED</td>
</tr>
<tr>
<td>0</td>
<td>MMDB_IS_NOT_ENABLED</td>
</tr>
<tr>
<td>-1</td>
<td>MMDB_INVALID_DATATYPE</td>
</tr>
</tbody>
</table>

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:

```c
#include <dmbimage.h>

rc = DBiIsColumnEnabled("employee",
    "picture", &status);
```
**DBilsFileReferenced**

**Table 130. The extenders that support DBilsFileReferenced**

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

**Table 131. The image files**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbimage.h

**Syntax**

```
long DBilsFileReferenced(
    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:

```c
#include <dmbimage.h>
long idx;

rc = DBiIsFileReferenced(NULL,
                "/images/ajones.bmp",
                &count, &tableList);
```
**DBIsServerEnabled**

**Table 132. The extenders that support **DBIsServerEnabled**

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

**Table 133. The image files**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbimage.h

**Syntax**

```c
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:

```c
#include <dmbimage.h>

rc = DBIsServerEnabled(&status);
```
DBIsTableEnabled

Table 134. The extenders that support DBIsTableEnabled

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 135. The image files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

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**

**Table 136. The extenders that support DBiPrepareAttrs**

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 z/OS 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**

**Table 137. The image files**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBIMAGE</td>
<td>libdmbimage.a</td>
<td>dmbimage.lib</td>
<td>libdmbimage.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbimage.h

**Syntax**

```c
void DBiPrepareAttrs(
    MMDBImageAttrs *imgAttr
);
```

**Parameters**

`imgAttr (in)`

The user-supplied attributes of the image.

**Examples**

Prepare user-supplied image attributes:

```c
#include <dmbimage.h>

DBiPrepareAttrs(&imgattr);
```
DBvAdminGetInaccessibleFiles

DBvAdminGetInaccessibleFiles

Table 138. The extenders that support DBvAdminGetInaccessibleFiles

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 139. The video files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file

dmbvideo.h

Syntax

```c
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.
DBvAdminGetInaccessibleFiles

**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 video columns of tables that are owned by user ID rsmith:

```c
#include <dmbvideo.h>

long idx;

rc = DBvAdminGetInaccessibleFiles
    ("rsmith", &count,
     &filelist);
```

Chapter 14. Application programming interfaces  285
DBvAdminGetReferencedFiles

Table 140. The extenders that support DBvAdminGetReferencedFiles

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 141. The video files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file

dmbvideo.h

Syntax

```c
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.

**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:

```c
#include <dmbvideo.h>
long idx;

rc = DBvAdminGetReferencedFiles
    ("ajones", &count,
     &filelist);
```
DBvAdminIsFileReferenced

Table 142. The extenders that support DBvAdminIsFileReferenced

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 143. The video files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file

dmbvideo.h

Syntax

```c
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.

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);
```
The extenders that support DBvDisableColumn

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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 CREATE privilege on the target database or is the DBADM for the database

Library file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file
dmbvideo.h

Syntax

```c
long DBvDisableColumn(
    char *tableName,
    char *colName,
);
```

Parameters

- **tableName (in)**
  
  The name of the table that contains the video column.

- **colName (in)**
  
  The name of the video column.
Error codes

**MMDB_SUCCESS**
API call processed successfully.

**MMDB_RC_NO_AUTH**
Caller does not have the proper access authority.

**MMDB_RC_NOT_CONNECTED**
Application does not have valid connection to a database server.

Examples

Disable the tv_ads column in the employee table for video (DB2Video data):

```c
#include <dmbvideo.h>

rc = DBvDisableColumn("employee",
                      "tv_ads");
```
Table 146. The extenders that support DBvDisableServer

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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**

Table 147. The video files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbvideo.h

**Syntax**

```c
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):

```c
#include <dmbvideo.h>

rc = DBvDisableServer();
```
**DBvDisableTable**

*Table 148. The extenders that support DBvDisableTable*

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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 CREATETAB privilege on the target database or is the DBADM for the database

**Library file**

*Table 149. The video files*

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbvideo.h

**Syntax**

```c
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.
**DBvDisableTable**

**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**

*Table 150. The extenders that support DBvEnableColumn*

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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**

*Table 151. The video files*

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbvideo.h

**Syntax**

```c
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.
**DBvEnableColumn**

- **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:

```c
#include <dmvvideo.h>

rc = DBvEnableColumn("employee",
            "video");
```
DBvEnableServer

Table 152. The extenders that support DBvEnableServer

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 153. The video files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file
dmbvideo.h

Syntax

```c
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 gbpcache-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 z/OS 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**

*Table 154. The extenders that support DBvEnableTable*

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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**

*Table 155. The video files*

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbvideo.h

**Syntax**

```c
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
**DBvEnableTable**

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:

```c
#include <dmbvideo.h>

rc = DBvEnableTable("myts,,mylobts", "employee");
```
DBvGetError

Table 156. The extenders that support DBvGetError

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Returns a description of the last error. Call this API after any other API returns an error code.

Authorization

None.

Library file

Table 157. The video files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file

dmbvideo.h

Syntax

```c
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`:

```c
#include <dmbvideo.h>

int rc = DBvGetError(&errCode, &errMsg);
```
DBvGetInaccessibleFiles

Table 158. The extenders that support DBvGetInaccessibleFiles

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 159. The video files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file

dmbvideo.h

Syntax

```c
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.

**Examples**

List all inaccessible files referenced in video columns in the employee table:

```c
#include <dmbvideo.h>
long idx;

rc = DBvGetInaccessibleFiles("employee",
                             &count, &filelist);
```
DBvGetReferencedFiles

Table 160. The extenders that support DBvGetReferencedFiles

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 161. The video files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file

dmbvideo.h

Syntax

```c
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.
**Examples**

List all files that are referenced in video columns in the employee table:
```c
#include <dmbvideo.h>  
long idx;  
rc = DBvGetReferencedFiles("employee",  
&count, &filelist);
```
DBvIsColumnEnabled

**Table 162. The extenders that support DBvIsColumnEnabled**

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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**

**Table 163. The video files**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbvideo.h

**Syntax**

```c
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:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MMDB_IS_ENABLED</td>
</tr>
<tr>
<td>0</td>
<td>MMDB_IS_NOT_ENABLED</td>
</tr>
<tr>
<td>-1</td>
<td>MMDB_INVALID_DATATYPE</td>
</tr>
</tbody>
</table>

**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 video column in the employee table is enabled for video:

```c
#include <dmbvideo.h>

rc = DBvIsColumnEnabled("employee",
                        "video", &status);
```
**DBvIsFileReferenced**

Table 164. The extenders that support DBvIsFileReferenced

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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**

Table 165. The video files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbvideo.h

**Syntax**

```c
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.

**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:

```c
#include <dmbvideo.h>
long idx;

rc = DBvIsFileReferenced(NULL,  
    "/videos/ajones.mpg",  
    &count, &tableList);
```
DBvIsServerEnabled

Table 166. The extenders that support DBvIsServerEnabled

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 167. The video files

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file

dmbvideo.h

Syntax

```c
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:

```c
#include <dmbvideo.h>

rc = DBvIsServerEnabled(&status);
```
**DBvIsTableEnabled**

*Table 168. The extenders that support DBvIsTableEnabled*

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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**

*Table 169. The video files*

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbvideo.h

**Syntax**

```c
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MMDB_IS_ENABLED</td>
</tr>
<tr>
<td>0</td>
<td>MMDB_IS_NOT_ENABLED</td>
</tr>
</tbody>
</table>

**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:
DBvisTableEnabled

#include <dmbvideo.h>

rc = DBvisTableEnabled("employee",
   &status);
Table 170. The extenders that support DBvPlay

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 171. The video files

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBVIDEO</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

Include file

dmbvideo.h

Syntax

Play a video stored in a column

```c
long DBvPlay(  
    char *playerName,  
    MMDB_PLAY_HANDLE,  
    DB2Video *videoHandle,  
    waitFlag  
);
```

Play a video stored as a file
long DBvPlay(
    char *playerName,
    MMDB_PLAY_HANDLE,  
    char *fileName,
    waitFlag
);

Parameters

playerName (in)
The name of the video player. If set to NULL, the default video player specified by the DB2VIDEOPAYER 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:

```c
#include <dmbvideo.h>

rc = DBvPlay(NULL, MMDB_PLAY_HANDLE, videoHandle,
             MMDB_PLAY_WAIT);
```
**DBvPrepareAttrs**

Table 172. The extenders that support DBvPrepareAttrs

<table>
<thead>
<tr>
<th></th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

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 z/OS 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**

Table 173. The video files

<table>
<thead>
<tr>
<th></th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>libdmbvideo.a</td>
<td>dmbvideo.lib</td>
<td>libdmbvideo.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbvideo.h

**Syntax**

```c
void DBvPrepareAttrs(
    MMDBVideoAttrs *vidAttr
);
```

**Parameters**

`vidAttr (in)`

The user-supplied attributes of the video.

**Examples**

Prepare user-supplied video attributes:

```c
#include <dmbvideo.h>

DBvPrepareAttrs(&vidattr);
```
QbAddFeature

Table 174. The extenders that support QbAddFeature

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 175. The image API file

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapilib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

Include file
dmbqbapi.h

Syntax

```c
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
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.

qbicECInvalidFeatureClass  
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

### Table 176. The extenders that support QbCatalogColumn

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**Table 177. The image API file**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

### Include file

dmbqbapi.h

### Syntax

```c
SQLRETURN QbCatalogColumn(
    QbCatalogHandle cHdl
);
```

### Parameters

- **cHdl (in)**
  
  A pointer to the handle of the catalog.

### Error codes

- **qbicECIvalidHandle**
  
  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.

qbicECSQLException
   An SQL error occurred.

Examples

#include <dmbqbapi.h>

rc = QbCatalogColumn(CatHdl);
QbCloseCatalog

**Table 178. The extenders that support QbCloseCatalog**

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Closes the catalog. The API frees the opened catalog handle and the allocated resources.

**Authorization**

None

**Library file**

**Table 179. The image API file**

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbqbapi.h

**Syntax**

```c
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`:

```c
#include <dmbqbapi.h>

rc = QbCloseCatalog(CatHdl);
```
QbCreateCatalog

Table 180. The extenders that support QbCreateCatalog

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 CREATEAB 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

Table 181. The image API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

Include file
dmbqbapi.h

Syntax

```c
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, 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:

```c
#include <dmbqbapi.h>

rc = QbCreateCatalog("employee",
    "picture", 1);
```

QbCreateCatalog
QbDeleteCatalog

Table 182. The extenders that support QbDeleteCatalog

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 183. The image API file

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

Include file

dmbqbapi.h

Syntax

```c
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

qbicECivalidHandle
The catalog handle is not valid.

qbicECCatalogInUse
The catalog was being used by someone else.

qbicECCatalogRO
The catalog is read-only.
QbDeleteCatalog

qbicECSys
A system error occurred.

qbicECSqlError
An SQL error occurred.

Examples

Delete the QBIC catalog associated with the picture column in the employee table:

#include <dmbqbapi.h>

rc=QbDeleteCatalog("employee", "picture");
Table 184. The extenders that support QbGetCatalogInfo

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

Table 185. The image API file

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbqbapi.h

**Syntax**

```c
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`:
QbGetCatalogInfo

#include <dmbqbapi.h>

rc = QbGetCatalogInfo(CatHdl, &catInfo);
QbListFeatures

Table 186. The extenders that support QbListFeatures

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 187. The image API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

Include file
dmbqbapi.h

Syntax

```c
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

- **qbicECIvalidHandle**
  The catalog handle is not valid.
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:

```c
#include <dmbqbapi.h>
rc = QbGetCatalogInfo(CatHdl, &catInfo);
bufSize = catInfo.featureCount*qbiMaxFeatureName;
rc = QbListFeatures(CatHdl, bufSize, count, featureNames);
```
QbOpenCatalog

Table 188. The extenders that support QbOpenCatalog

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 189. The image API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapl.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

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.
QbOpenCatalog

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:

```c
#include <dmbqbapi.h>

rc=QbOpenCatalog("employee", "picture",
qbiread, &CatHdl);
```
QbQueryAddFeature

Table 190. The extenders that support QbQueryAddFeature

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adds the specified feature to a QBIC Catalog.

Authorization

None.

Library file

Table 191. The image query API file

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

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.
QbQueryAddFeature

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

**Table 192. The extenders that support QbQueryCreate**

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

**Table 193. The image query API file**

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBQBAPI</td>
<td>libdmbqbpapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbpapi.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbqbapi.h

**Syntax**

```c
SQLRETURN QbQueryCreate(
    QbQueryHandle *qObj
);
```

**Parameters**

**qObj (out)**

A pointer to the query handle. If unsuccessful, this handle is set to 0.

**Error codes**

**qbiECallocation**

The system cannot allocate enough memory.

**Examples**

Create a query object and return the handle in qoHandle:

```c
#include <dmbqbapi.h>

rc = QbQueryCreate(&qoHandle);
```
QbQueryDelete

Table 194. The extenders that support QbQueryDelete

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Deletes an unnamed query object. The API releases all the memory used by the query object and any added features.

Authorization

None.

Library file

Table 195. The query API file

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
<td></td>
</tr>
</tbody>
</table>

Include file

dmbqbapi.h

Syntax

```c
SQLRETURN QbQueryDelete(
    QbQueryHandle qObj
);
```

Parameters

qObj (in)
The handle of the query object.

Error codes

qbiECinvalidQueryHandle
The query object handle you specified does not reference a valid query.

Examples

Delete the query object identified by the handle qoHandle:
```c
#include <dmbqbapi.h>
rc = QbQueryDelete(qoHandle);
```
QbQueryGetFeatureCount

Table 196. The extenders that support QbQueryGetFeatureCount

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 197. The query API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

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:

```c
#include <dmbqbapi.h>

rc = QbQueryGetFeatureCount(qoHandle, &count);
```
QbQueryGetString

Table 198. The extenders that support QbQueryGetString

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 199. The query API file

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

Include file

dmbqbapi.h

Syntax

```sql
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.

```c
#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;
}
```

336  Image, Audio, and Video Extenders Administration and Programming
QbQueryListFeatures

Table 200. The extenders that support QbQueryListFeatures

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 201. The query API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

Include file

dmbqbapi.h

Syntax

```c
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.
QbQueryListFeatures

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:

```c
#include <dmbqbapi.h>

bufSize = qbiMaxFeatureName;
rc = QbQueryListFeatures(qoHandle, bufSize,
                          &retCount, feats);
```
QbQueryNameCreate

Table 202. The extenders that support QbQueryNameCreate

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 203. The query API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

Include file

dmbqbapi.h

Syntax

```c
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:

```c
#include <dmbqbapi.h>

rc = QbQueryNameCreate(qHandle,
    "fshavgcol",
    "average color query, 10/15/96");
```
# QbQueryNameDelete

Table 204. The extenders that support QbQueryNameDelete

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

Table 205. The query API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbqbapi.h

**Syntax**

```c
SQLRETURN QbQueryNameDelete(
    char *name
);
```

**Parameters**

- **name (in)**
  
  The name of the query object you are deleting.

**Error codes**

- **qbiECinvalidQueryHandle**
  
  The query object handle that you specified does not reference a valid query object.

**Examples**

Delete the query object named fshavgcol:

```c
#include <dmbqbapi.h>

rc = QbQueryNameDelete("fshavgcol");
```
QbQueryNameSearch

*Table 206. The extenders that support QbQueryNameSearch*

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

*Table 207. The query API file*

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
<td></td>
</tr>
</tbody>
</table>

**Include file**

dmbqbapi.h

**Syntax**

```c
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.
column Name (in)
The name of the image column. The column must be enabled for image data.

max Returns (in)
The maximum number of images you want returned.

scope (in) (Reserved)
Must be set to 0 (NULL)

result Type (in) (Reserved)
Must be set to qbiArray.

count (out)
The pointer to the number of images returned. If zero is returned, make sure the image column is cataloged for all the features in the query object.

returns (out)
The pointer to the array of QbResult structures that hold the returned results. Make sure you allocate the buffer large enough to hold all the results you expect.

Error codes
qbiECinvalidQueryHandle
The query object handle you specified does not reference a valid query object.

Examples
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:

```c
#include <dmbqabapi.h>

rc = QbQueryNameSearch("fshavgcol",
    "employee", "picture",
    6, 0, qbiArray, &count, &returns);
```
QbQueryRemoveFeature

QbQueryRemoveFeature

Table 208. The extenders that support QbQueryRemoveFeature

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 209. The query API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

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:

```c
#include <dmbqbapi.h>

rc = QbQueryRemoveFeature(qoHandle,
   "QbColorFeatureClass");
```
QbQuerySearch

QbQuerySearch

Table 210. The extenders that support QbQuerySearch

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 211. The query API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

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.

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:

```c
#include <dmbqbapi.h>

rc = QbQuerySearch(qHandle, "employee",
    "picture", 6, 0, qbiArray,
    &count, &returns);
```
QbQuerySetFeatureData

**Table 212. The extenders that support QbQuerySetFeatureData**

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

**Table 213. The query API file**

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

**Include file**

dmbqbapi.h

**Syntax**

```c
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 126 for more information.
Error codes

qbiECinvalidQueryHandle
   The query object handle you specified does not reference a valid query object.

qbiECunknownFeatureClass
   The feature you specified is not a recognized feature class name.

qbiECinvalidFeatureClass
   The feature you specified is not a valid name format.

qbiECfeatureNotPresent
   The feature you specified is not a member of the query object.

qbiECfileUnreadable
   The image source file cannot be found or read.

Examples

Set the data source for the histogram color feature in a query object. The data source for the feature is an image in a users table:

```c
#include <dmbqbapi.h>

QbQueryHandle qoHandle;
QbImageSource imgSource;

imgSource.sourceType = qbiSource_ImageHandle;
strcpy(imgSource.imageHandle,handleFile);

rc = QbQuerySetFeatureData(qoHandle, "QbColorHistogramFeatureClass", &imgSource);
```
QbQuerySetFeatureWeight

Table 214. The extenders that support QbQuerySetFeatureWeight

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sets the weight of the specified feature in a query object.

Authorization

None.

Library file

Table 215. The query API file

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td></td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

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:

```c
#include <dmbqbapi.h>

weight=2.0
rc = QbQuerySetFeatureWeight(qoHandle, "QbColorFeatureClass", &weight);
```
QbQueryStringSearch

Table 216. The extenders that support QbQueryStringSearch

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 217. The query API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

Include file

dmbqbapi.h

Syntax

```c
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).
QbQueryStringSearch

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

Table 218. The extenders that support QbReCatalogColumn

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Audio</td>
<td>Video</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 CREATEAB privilege on the target database or is the DBADM for the database

Library file

Table 219. The image API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

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.
QbReCatalogColumn

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(CatHd1);
QbRemoveFeature

Table 220. The extenders that support QbRemoveFeature

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 221. The image API file

<table>
<thead>
<tr>
<th>z/OS</th>
<th>AIX</th>
<th>Windows</th>
<th>Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMBQBAPI</td>
<td>libdmbqbapi.a</td>
<td>dmbqbapi.lib</td>
<td>libdmbqbapi.so</td>
</tr>
</tbody>
</table>

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

qbicECIvalidHandle
   The catalog handle is not valid.

qbicECCatalogReadOnly
   The catalog is open for read only.

qbicECFeatureNotFound
   The feature is not in the catalog.

qbiECInvalidFeatureClass
   The feature you specified is not a valid name format.

Examples

Remove the QbColorHistogramFeatureClass feature from the catalog identified by the handle CatHdl:
QbRemoveFeature

#include <dmbqbapi.h>

rc=QbRemoveFeature(CatHd1,
                   "QbColorHistogramFeatureClass");
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:

Table 222. Entering DB2 Extender administration commands

<table>
<thead>
<tr>
<th>Client</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX, Solaris</td>
<td>Enter the DB2EXT command from the operating system command prompt.</td>
</tr>
<tr>
<td>Windows XP, Windows 2000, Windows NT</td>
<td>Double-click on the DB2EXT Command Line Processor icon in the DB2 Extenders folder, or enter the DB2EXT command from the DB2 command window.</td>
</tr>
<tr>
<td>OpenEdition</td>
<td>Issue the command:</td>
</tr>
<tr>
<td></td>
<td>db2ext -a subsystem_ID</td>
</tr>
<tr>
<td></td>
<td>where subsystem_ID is the identification of the pertinent DB2 subsystem (for example, V61A).</td>
</tr>
<tr>
<td>TSO</td>
<td>Issue the TSO call:</td>
</tr>
<tr>
<td></td>
<td>call 'loadlib' '-a subsystem_ID' asis</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
</tbody>
</table>

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

Table 223. The extenders that support the ADD QBIC FEATURE

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 224. The extenders that support CATALOG QBIC COLUMN

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```
CATALOG QBIC COLUMN FOR NEW ALL
```

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

Table 225. The extenders that support CLOSE QBIC CATALOG

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Audio</td>
<td>Video</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 226. The extenders that support CREATE QBIC CATALOG

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
CREATE QBIC CATALOG

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

DELETE QBIC CATALOG

Table 227. The extenders that support DELETE QBIC CATALOG

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 228. The extenders that support DISABLE COLUMN

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

```bash
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

Table 229. The extenders that support DISABLE SERVER

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 230. The extenders that support DISABLE TABLE

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

Table 231. The extenders that support ENABLE COLUMN

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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 CREATE 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**

---

*Table 232. The extenders that support ENABLE SERVER*

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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**

```
ENABLE SERVER FOR extender_name USING tablespace_name
```

```
WLM ENVIRONMENT wlm_names
```

```
EXTERNAL SECURITY external_security
```

**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, 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 z/OS 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

Table 233. The extenders that support ENABLE TABLE

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

\[\text{ENABLE TABLE–table_name–FOR–extender_name–USING–tablespace_name}\]

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.
ENABLE TABLE

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 employee for db2image using myts,,mylobts

Usage notes
Connect to the database server before using this command.
GET EXTENDER STATUS

**Table 234. The extenders that support GET EXTENDER STATUS**

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Displays the names of the extenders, if any, for which a column, table, or the current database server is enabled.

**Authorization**

None

**Command syntax**

```
GET EXTENDER STATUS
  IN table_name
  COLUMN table_name.col_name
```

**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

Table 235. The extenders that support GET INACCESSIBLE FILES

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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

```
GET INACCESSIBLE FILES
          FOR extender_name
          USER user_ID
          IN table_name
```

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 that qualifier. If you specify neither, the command lists inaccessible files for all the tables in the current database server.
GET QBIC CATALOG INFO

GET QBIC CATALOG INFO

Table 236. The extenders that support GET QBIC CATALOG INFO

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 237. The extenders that support GET REFERENCED FILES

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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, SYSCMAINT, DBADM

For a particular table (if you specify IN) or tables that belong to a qualifier (if you specify USER): Select

Command syntax

```plaintext
GET REFERENCED FILES 
   FOR extender_name
   USER user_ID
   IN table_name
```

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

Table 238. The extenders that support GRANT

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Grants privileges on administrative support tables to users, including privileges on administrative support tables for QBIC catalogs.

Authorization

SYSADM, DBADM

Command syntax

```
GRANT [SELECT, UPDATE] FOR [ALL | extender_name] ON table_name TO [PUBLIC | USER] user_ID
```

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 select for db2image on employee to user ajones
```
GRANT

Usage notes
OPEN QBIC CATALOG

Table 239. The extenders that support OPEN QBIC CATALOG

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 240. The extenders that support QUIT

<table>
<thead>
<tr>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Shuts down the db2ext command-line processor for command entry in interactive mode. For z/OS 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 z/OS clients, QUIT drops the connection to DB2. For workstation clients, QUIT maintains the connection to DB2.
## REMOVE QBIC FEATURE

### Table 241. The extenders that support REMOVE QBIC FEATURE

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Deletes the feature table of the specified feature from the opened catalog.

### Authorization

Alter, Control, SYSADM, DBADM

### Command syntax

```plaintext
RM 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:

```plaintext
remove qbic feature qbcolorfeatureclass
```

### Usage notes

Connect to the database before using this command.

The catalog must be open.
REVOKE

Table 242. The extenders that support REVOKE

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Revoke privileges on administrative support tables to users, including privileges on administrative support tables for QBIC catalogs.

Authorization

SYSADM, DBADM

Command syntax

```
REVOKE SELECT FOR ALL extender_name ON table_name FROM PUBLIC
```

Command parameters

**SELECT**

Revoke SELECT privilege on all administrative support tables associated with the user table for the specified extenders. If the Image Extender is specified (DB2Image), also revoke SELECT privilege on the QBIC catalog tables associated with the user table.

**UPDATE**

Revoke INSERT, UPDATE, and DELETE privileges on the administrative support tables for the QBIC catalogs associated with the user table.

**ALL**

Revoke 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**

Revoke 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**

Table 243. The extenders that support TERMINATE

<table>
<thead>
<tr>
<th></th>
<th>Image</th>
<th>Audio</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

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 SQLLError function. (For more information about retrieving error information with the SQLLError 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 388 for a description of each SQLSTATE code returned by the DB2 Extenders.

Embedded SQL statements and DB2 CLI calls that invoke DB2 extender UDFs might return SQLCODE messages and SQLSTATE values that are unique to these UDFs, but DB2 returns these values in the same way as it does for other...
Handling UDF codes

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 392 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 219 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 392 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 244 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 392 for information about each message.

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td></td>
<td>MMDB_SQLSTATE_OK</td>
</tr>
<tr>
<td>01H01</td>
<td>DMB0211W</td>
<td>MMDB_SQLSTATE_WARN_NO_OVERWRITE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The file overwrite does not take place</td>
</tr>
<tr>
<td>38A00</td>
<td>DMB0101E</td>
<td>MMDB_SQLSTATE_AUDIO_NULL_PARM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Input parameter to the UDF cannot be null</td>
</tr>
<tr>
<td>38A02</td>
<td>DMB0209E</td>
<td>MMDB_SQLSTATE_AUDIO_OPEN_HDR_ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error occurred while opening audio file header</td>
</tr>
<tr>
<td>38A03</td>
<td>DMB0209E</td>
<td>MMDB_SQLSTATE_AUDIO_BAD_WAVE_HDR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Invalid wave file supplied</td>
</tr>
<tr>
<td>38V00</td>
<td>DMB0101E</td>
<td>MMDB_SQLSTATE_VIDEO_NULL_PARM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Input parameter to the UDF cannot be null</td>
</tr>
</tbody>
</table>
### Table 244. SQLSTATE codes and associated message numbers (continued)

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message No.</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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    | MMDB_SQLSTATE_INVALID_INPUT  
Input parameter to the UDF is not valid |
|          | DMB0101E    | MMDB_SQLSTATE_INVALID_INPUT  
Input parameter to the UDF is not valid |
|          | DMB0102E    | MMDB_SQLSTATE_INVALID_INPUT  
Input parameter to the UDF is not valid |
|          | DMB0103E    | MMDB_SQLSTATE_INVALID_INPUT  
Input parameter to the UDF is not valid |
|          | 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    | MMDB_SQLSTATE_FREE_STMT_FAIL  
Error occurred while freeing statement |
| 38619    | DMB0208E    | 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    | MMDB_SQLSTATE_SET_PARAM_FAIL  
Error occurred while setting parameter |
| 38623    | DMB0208E    | MMDB_SQLSTATE_PREPARE_FAIL  
Error occurred while preparing an SQL statement |
Table 244. SQLSTATE codes and associated message numbers (continued)

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>38624</td>
<td>DMB0208E</td>
<td>MMDB_SQLSTATE_EXECUTE_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while executing a statement</td>
</tr>
<tr>
<td></td>
<td>DMB0172E</td>
<td></td>
</tr>
<tr>
<td>38625</td>
<td>DMB0208E</td>
<td>MMDB_SQLSTATE_EXEC_DIRECT_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while directly executing SQL statement in UDF</td>
</tr>
<tr>
<td>38626</td>
<td>DMB0208E</td>
<td>MMDB_SQLSTATE_FETCH_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while retrieving next row of data</td>
</tr>
<tr>
<td>38627</td>
<td>DMB0208E</td>
<td>MMDB_SQLSTATE_COMMIT_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while committing the transaction</td>
</tr>
<tr>
<td>38628</td>
<td>DMB0208E</td>
<td>MMDB_SQLSTATE_GET_LENGTH_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while retrieving the length of a string value</td>
</tr>
<tr>
<td>38629</td>
<td>DMB0208E</td>
<td>MMDB_SQLSTATE_GET_SUBSTRING_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while retrieving a portion of a string value</td>
</tr>
<tr>
<td>38650</td>
<td>DMB0077E</td>
<td>MMDB_SQLSTATE_COPY_BLOB_2_FILE_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0079E</td>
<td>Error occurred while copying BLOB to a file</td>
</tr>
<tr>
<td>38651</td>
<td>DMB0086E</td>
<td>MMDB_SQLSTATE_BLOB_BUFFER_TOO_SMALL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>The buffer supplied is too small</td>
</tr>
<tr>
<td>38652</td>
<td>DMB0082E</td>
<td>MMDB_SQLSTATE_BUILD_HANDLE</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while constructing media column data</td>
</tr>
<tr>
<td>38653</td>
<td>DMB0083E</td>
<td>MMDB_SQLSTATE_INVALID_INSERT_VIA_SELECT</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>The request for insert via select is not valid</td>
</tr>
<tr>
<td>38654</td>
<td>DMB0081E</td>
<td>MMDB_SQLSTATE_INVALID_OFFSET_SIZE</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>The offset size is not valid</td>
</tr>
<tr>
<td>38655</td>
<td>DMB0068E</td>
<td>MMDB_SQLSTATE_METATABLE_DOESNOT_EXIST</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>The required metadata table does not exist</td>
</tr>
<tr>
<td>38670</td>
<td>DMB0134E</td>
<td>MMDB_SQLSTATE_UNKNOWN_FORMAT</td>
</tr>
<tr>
<td></td>
<td>DMB0103E</td>
<td>The stored media has unknown format</td>
</tr>
<tr>
<td>38671</td>
<td>DMB0135E</td>
<td>MMDB_SQLSTATE_CREATE_THUMBNAIL_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while creating the thumbnail</td>
</tr>
<tr>
<td>38672</td>
<td>DMB0114E</td>
<td>MMDB_SQLSTATE_FORMAT_CONVERSION_FAIL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred while converting the file format</td>
</tr>
<tr>
<td>38673</td>
<td>DMB0363E</td>
<td>MMDB_SQLSTATE_INVALID_UPDATE</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred when an update UDF was invoked without referencing a table</td>
</tr>
<tr>
<td>38674</td>
<td>DMB0361E</td>
<td>MMDB_SQLSTATE_NOT_ENABLED</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Error occurred when an import UDF was applied to a column which was not enabled for the extender</td>
</tr>
<tr>
<td>38675</td>
<td>DMB0129E</td>
<td>MMDB_SQLSTATE_VIDEO_INTERNAL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Internal error in Video Extender UDFs</td>
</tr>
<tr>
<td>38676</td>
<td>DMB0129E</td>
<td>MMDB_SQLSTATE_AUDIO_INTERNAL</td>
</tr>
<tr>
<td></td>
<td>DMB0133E</td>
<td>Internal error in Audio Extender UDFs</td>
</tr>
<tr>
<td>38677</td>
<td>DMB0129E</td>
<td>MMDB_SQLSTATE_IMAGE_INTERNAL</td>
</tr>
<tr>
<td>38678</td>
<td>DMB0089E</td>
<td>MMDB_SQLSTATE_BASE_INTERNAL_ERROR</td>
</tr>
<tr>
<td></td>
<td>DMB0208E</td>
<td>Internal error in common layer</td>
</tr>
</tbody>
</table>
Table 244. SQLSTATE codes and associated message numbers (continued)

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>38681</td>
<td>DMB0108E</td>
<td>MMDB_SQLSTATE_IMPORT_ENV_NOT_SETUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Environment variable for import is not set up correctly</td>
</tr>
<tr>
<td>38682</td>
<td>DMB0111E</td>
<td>MMDB_SQLSTATE_STORE_ENV_NOT_SETUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Environment variable for store operation is not set up correctly</td>
</tr>
<tr>
<td>38683</td>
<td>DMB0107E</td>
<td>MMDB_SQLSTATE_EXPORT_ENV_NOT_SETUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Environment variable for export operation is not set up correctly</td>
</tr>
<tr>
<td>38684</td>
<td>DMB0117E</td>
<td>MMDB_SQLSTATE_TEMP_ENV_NOT_SETUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Environment variable for creating temporary files is not set up correctly</td>
</tr>
<tr>
<td>38686</td>
<td>DMB0109E</td>
<td>MMDB_SQLSTATE_CANT_RESOLVE_IMPORT_FILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error occurred while resolving import file name</td>
</tr>
<tr>
<td>38687</td>
<td>DMB0112E</td>
<td>MMDB_SQLSTATE_CANT_RESOLVE_STORE_FILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error occurred while resolving store file name</td>
</tr>
<tr>
<td>38688</td>
<td>DMB0110E</td>
<td>MMDB_SQLSTATE_CANT_RESOLVE_EXPORT_FILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error occurred while resolving export file name</td>
</tr>
<tr>
<td>38689</td>
<td>DMB0116E</td>
<td>MMDB_SQLSTATE_CANT_CREATE_TMP_FILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Error occurred while creating temporary file</td>
</tr>
<tr>
<td>38690</td>
<td>DMB0076E</td>
<td>MMDB_SQLSTATE_OPEN_IMPORT_FILE_FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cannot open import file</td>
</tr>
<tr>
<td>38691</td>
<td>DMB0115E</td>
<td>MMDB_SQLSTATE_OPEN_STORE_FILE_FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cannot open import file</td>
</tr>
<tr>
<td>38692</td>
<td>DMB0114E</td>
<td>MMDB_SQLSTATE_OPEN_EXPORT_FILE_FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cannot open export file</td>
</tr>
<tr>
<td>38693</td>
<td>DMB0118E</td>
<td>MMDB_SQLSTATE_OPEN_TEMP_FILE_FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cannot open temporary file</td>
</tr>
<tr>
<td>38694</td>
<td>DMB0117E</td>
<td>MMDB_SQLSTATE_OPEN_CONTENT_FILE_FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cannot open content file</td>
</tr>
<tr>
<td>38695</td>
<td>DMB0135E</td>
<td>MMDB_SQLSTATE_OPEN_THUMBNAIL_FILE_FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cannot open thumbnail file</td>
</tr>
<tr>
<td>38696</td>
<td>DMB0135E</td>
<td>MMDB_SQLSTATE_READ_THUMBNAIL_FILE_FAIL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cannot read thumbnail file</td>
</tr>
<tr>
<td>38697</td>
<td>DMB0207E</td>
<td>MMDB_SQLSTATE_OVERWRITE_NOT_ALLOWED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The overwrite operation could not be performed</td>
</tr>
<tr>
<td>38699</td>
<td>DMB0171E</td>
<td>MMDB_SQLSTATE_QUERY_NAME_NOT_FOUND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A query with that name was not found</td>
</tr>
<tr>
<td>38700</td>
<td></td>
<td>MMDB_SQLSTATE_NO_MANAGEBLOB</td>
</tr>
<tr>
<td>38701</td>
<td></td>
<td>MMDB_SQLSTATE_UDFLOCATOR_FAIL</td>
</tr>
<tr>
<td>38702</td>
<td></td>
<td>MMDB_SQLSTATE_SQL_FAIL</td>
</tr>
<tr>
<td>38703</td>
<td></td>
<td>MMDB_SQLSTATE_INVALID_UPDATE</td>
</tr>
<tr>
<td>38704</td>
<td></td>
<td>MMDB_SQLSTATE_NOT_ENABLED</td>
</tr>
<tr>
<td>38705</td>
<td>DMB0366E</td>
<td>MMDB_SQLSTATE_QBIC_QUERY_FAIL_TO_BUILD</td>
</tr>
<tr>
<td></td>
<td>DMB0382E</td>
<td>A failure occurred in building the query</td>
</tr>
</tbody>
</table>
### SQLSTATEs

#### Table 244. SQLSTATE codes and associated message numbers (continued)

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>38706</td>
<td>DMB0205E</td>
<td>MMDB_SQLSTATE_QBIC_TABLE_COLUMN_PAIR_NOT_VALID&lt;br&gt;A failure occurred when trying to access a QBIC catalog.&lt;br&gt;Either an image handle wasn’t found in the catalog, or&lt;br&gt;the combination of the table name and column name&lt;br&gt;does not have a catalog.</td>
</tr>
<tr>
<td>38707</td>
<td>DMB0383E</td>
<td>MMDB_SQLSTATE_QBIC_QUERY_EXECUTE_FAILED&lt;br&gt;A failure occurred in running the query</td>
</tr>
<tr>
<td>38708</td>
<td>DMB0208E</td>
<td>MMDB_SQLSTATE_QBIC_UNKNOWN_ERROR&lt;br&gt;An unknown failure occurred in QBIC</td>
</tr>
<tr>
<td>38709</td>
<td>DMB0209E</td>
<td>MMDB_COPY_FILE_TO_LOCATOR_FAILURE&lt;br&gt;A failure occurred in copying a file to a LOB locator</td>
</tr>
<tr>
<td>38710</td>
<td>DMB0534E</td>
<td>MMDB_SQLSTATE_QBIC_UNSUPPORTED_UDF&lt;br&gt;The UDF is not supported.</td>
</tr>
</tbody>
</table>

### 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.

- **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.

- **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 refered to in table "<table-names>" 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 authority, 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 Extenders server cannot connect to the database or allocate a DB2 statement handle. The DB2 Extenders 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-names>" 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.

DMB0058E  "<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 reenable 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.
<table>
<thead>
<tr>
<th>Message ID</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMB0074E</td>
<td>USE statement handle not valid.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>An internal error has occurred.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Repeat the operation. If the error occurs again, contact IBM service.</td>
</tr>
<tr>
<td>DMB0075E</td>
<td>Specify a file name.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The operation requires a media file name.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Enter the name of a media file.</td>
</tr>
<tr>
<td>DMB0076E</td>
<td>Can’t open import file.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The import file is either missing or damaged.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Verify the name and existence of the import file.</td>
</tr>
<tr>
<td>DMB0077E</td>
<td>Can’t open/read content file.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The extender handle points to a file that does not exist or is corrupted. The file has become inaccessible to the extender.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Use the FILENAME UDF to find the name of the file, or verify the existence of the content file.</td>
</tr>
<tr>
<td>DMB0078E</td>
<td>Can’t create export file.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The export file is either missing or corrupted.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Verify the name and existence of the export file.</td>
</tr>
<tr>
<td>DMB0079E</td>
<td>Can’t copy BLOB to file.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The file cannot accept the BLOB. There might not be enough storage space to accommodate the BLOB.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Compare the size of the BLOB to the available storage, and increase storage if necessary.</td>
</tr>
<tr>
<td>DMB0080E</td>
<td>Can’t write to file.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The file is damaged or does not exist, or the name is misspelled.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Verify the name and existence of the file.</td>
</tr>
<tr>
<td>DMB0081E</td>
<td>Offset or size invalid.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The operation did not find the expected data in the data structure. Either the size of the field or the offset is incorrect.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Verify the offset and the size of the field.</td>
</tr>
<tr>
<td>DMB0082E</td>
<td>Can’t build handle.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>An internal error has occurred.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Repeat the operation. If the error occurs again, contact IBM service.</td>
</tr>
<tr>
<td>DMB0083E</td>
<td>“&lt;extender-name&gt;” and “&lt;extender-name&gt;” incompatible.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The two extenders specified in the message are not compatible in this usage. The insert operation, by either full or subselect, is not valid.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Make sure that your target object is enabled for the same extender for which the source object is enabled.</td>
</tr>
<tr>
<td>DMB0084E</td>
<td>Import request invalid: filename, content, storage type.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The import operation failed. Either the file name, the content, or the storage type was not valid.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Check the data and try again.</td>
</tr>
<tr>
<td>DMB0085E</td>
<td>The update request is not valid: filename, content, storage type.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The update operation failed. Either the file name, the content, or the storage type was not valid.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Check the data and try again.</td>
</tr>
<tr>
<td>DMB0086E</td>
<td>Requested size too large.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The size you requested is larger than the maximum blob size for the UDF.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Request a smaller size.</td>
</tr>
<tr>
<td>DMB0087E</td>
<td>File name invalid.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>There is no file with that name.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Verify the name and existence of the file.</td>
</tr>
<tr>
<td>DMB0088E</td>
<td>Handle value is NULL.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The UDF was expecting a non-null handle.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Make sure that the application gets a valid handle and passes it to the UDF.</td>
</tr>
<tr>
<td>DMB0089E</td>
<td>Handle value doesn’t exist.</td>
</tr>
<tr>
<td><strong>Cause:</strong></td>
<td>The handle passed to the UDF is not valid.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Make sure the application passes a valid handle.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DMB0090E</td>
<td>Data truncated.</td>
</tr>
<tr>
<td>DMB0091W</td>
<td>Content already in file.</td>
</tr>
<tr>
<td>DMB0092E</td>
<td>The insert operation attempted for column &quot;&lt;column-name&gt;&quot; is not valid.</td>
</tr>
<tr>
<td>DMB0093E</td>
<td>The update operation attempted for column &quot;&lt;column-name&gt;&quot; is not valid.</td>
</tr>
<tr>
<td>DMB0094I</td>
<td>Table &quot;&lt;table-name&gt;&quot; does not exist.</td>
</tr>
<tr>
<td>DMB0095W</td>
<td>The table &quot;&lt;table-name&gt;&quot; is not enabled for the &quot;&lt;extender-name&gt;&quot; extender.</td>
</tr>
<tr>
<td>DMB0096W</td>
<td>Column &quot;&lt;column-name&gt;&quot; in table &quot;&lt;table-name&gt;&quot; was not enabled for the &quot;&lt;extender-name&gt;&quot; extender.</td>
</tr>
<tr>
<td>DMB0097W</td>
<td>The current database is not enabled for the &quot;&lt;extender-name&gt;&quot; extender.</td>
</tr>
<tr>
<td>DMB0098E</td>
<td>The user does not have &quot;&lt;authority-level&gt;&quot; authority on table &quot;&lt;table-name&gt;&quot;.</td>
</tr>
<tr>
<td>DMB0099E</td>
<td>Can't commit transaction.</td>
</tr>
<tr>
<td>DMB0100E</td>
<td>&quot;&lt;name&gt;&quot; is not a valid table name.</td>
</tr>
<tr>
<td>DMB0101E</td>
<td>Invalid NULL parameter.</td>
</tr>
<tr>
<td>DMB0102E</td>
<td>Invalid storage type.</td>
</tr>
<tr>
<td>DMB0103E</td>
<td>Unsupported format.</td>
</tr>
</tbody>
</table>
Messages

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.

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 413.

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 413.

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 413.

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 413.

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 413.

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 413.

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 413.
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 DB Extenders," on page 413.

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.

DMB0123E  SQL set parameter failed.
Cause: An internal error has occurred.
Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0124E  SQL execute statement failed.
Cause: An internal error has occurred.
Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0125E  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.

DMB0126E  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.

DMB0127E  Can’t find bind file.
Cause: An internal error has occurred.
Action: Repeat the operation. If the error occurs again, contact IBM service.

DMB0128E  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.

DMB0129E  Can’t free an SQL statement.
Cause: An internal error has occurred.
Action: Repeat the operation. If the error occurs again, contact IBM service.

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
### Messages

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Cause</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMB0141E</td>
<td>The qualifier &quot;&lt;identifier&gt;&quot; that follows &quot;&lt;keyword&gt;&quot; is not valid.</td>
<td>The system cannot identify the qualifier in the command.</td>
<td>Check the qualifier and try again.</td>
</tr>
<tr>
<td>DMB0142E</td>
<td>No catalog was opened.</td>
<td>In the DB2 Extenders, the current command needs a QBIC catalog to be opened.</td>
<td>Open the QBIC catalog with the OPEN QBIC CATALOG command.</td>
</tr>
<tr>
<td>DMB0143I</td>
<td>In the QBIC catalog for column &quot;&lt;column-name&gt;&quot; in table &quot;&lt;table-name&gt;&quot;, auto-cataloging has been set to &quot;&lt;status&gt;&quot;. There are &quot;&lt;count&gt;&quot; features:</td>
<td>The message indicates the number of features defined in the QBIC catalog for a specific image column, and whether auto-cataloging is turned on.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0145E</td>
<td>The query handle is not valid.</td>
<td>The query handle you used in the API call is not valid.</td>
<td>Check your application to see if you are obtaining the correct query handle.</td>
</tr>
<tr>
<td>DMB0146E</td>
<td>The feature class name &quot;&lt;feature-class&gt;&quot; is not valid.</td>
<td>There is no feature class with that name. Valid feature names include:</td>
<td>Correct the name of the feature and try again.</td>
</tr>
<tr>
<td></td>
<td>- QbColorFeatureClass</td>
<td>• QbColorFeatureClass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- QbColorHistogramFeatureClass</td>
<td>• QbColorHistogramFeatureClass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- QbDrawFeatureClass</td>
<td>• QbDrawFeatureClass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- QbTextureFeatureClass</td>
<td>• QbTextureFeatureClass</td>
<td></td>
</tr>
<tr>
<td>DMB0147E</td>
<td>The feature class name &quot;&lt;feature-class&gt;&quot; is either missing or not valid.</td>
<td>Valid feature names include:</td>
<td>Correct the name of the feature and try again.</td>
</tr>
<tr>
<td></td>
<td>- QbColorFeatureClass</td>
<td>• QbColorFeatureClass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- QbColorHistogramFeatureClass</td>
<td>• QbColorHistogramFeatureClass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- QbDrawFeatureClass</td>
<td>• QbDrawFeatureClass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- QbTextureFeatureClass</td>
<td>• QbTextureFeatureClass</td>
<td></td>
</tr>
<tr>
<td>DMB0148E</td>
<td>Feature &quot;&lt;feature-name&gt;&quot; is already a member of the query.</td>
<td>The query already supports the feature indicated in the message.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0149E</td>
<td>Feature &quot;&lt;feature-name&gt;&quot; is not a member of the query.</td>
<td>The query does not include the specified feature name.</td>
<td>To add the feature to the query before calling other APIs that access the feature, use the QbQueryAddFeature API.</td>
</tr>
<tr>
<td>DMB0150E</td>
<td>The system cannot allocate memory.</td>
<td>The system was unable to allocate memory required to support the attempted operation.</td>
<td>Verify that your system has enough memory to complete the operation.</td>
</tr>
<tr>
<td>DMB0151E</td>
<td>The pointer to the return value is NULL.</td>
<td>The API call did not complete successfully because the pointer to a return value must not be NULL.</td>
<td>Make sure that valid parameters are supplied to the API call and the syntax is followed correctly.</td>
</tr>
<tr>
<td>DMB0152E</td>
<td>The pointer to the list return value is NULL.</td>
<td>The API call did not complete successfully because the pointer to a return value must not be NULL.</td>
<td>Make sure that valid parameters are supplied to the API call and the syntax is followed correctly.</td>
</tr>
<tr>
<td>DMB0153E</td>
<td>The scope parameter is reserved and must be 0.</td>
<td>The parameter is reserved for future use.</td>
<td>Set the scope to 0.</td>
</tr>
<tr>
<td>DMB0154E</td>
<td>The pointer to the feature class name is not valid.</td>
<td>The API call expected a valid pointer to the input feature class name.</td>
<td>Make sure that valid parameters are supplied to the API call and the syntax is followed correctly.</td>
</tr>
<tr>
<td>Message Code</td>
<td>Message Description</td>
<td>Cause</td>
<td>Action</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DMB0155I</td>
<td>A buffer size of zero was passed to the &quot;&lt;function-name&gt;&quot; function.</td>
<td>The API call needs the buffer to return information.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0156E</td>
<td>The QbImageSource pointer is NULL.</td>
<td>A NULL value indicates that the structure should not be changed.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0157E</td>
<td>The QbImageSource type &quot;&lt;type&gt;&quot; is not valid.</td>
<td>The data type of the structure should be QbImageSource.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0159E</td>
<td>The pointer to the QbImageSource image buffer is NULL.</td>
<td>The API call expected a pointer to be returned.</td>
<td>Check your application to see if the API call and the buffer is specified correctly.</td>
</tr>
<tr>
<td>DMB0160I</td>
<td>The length of the image buffer or file is zero.</td>
<td>The length is zero.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0161E</td>
<td>The pointer to the table and/or column name is NULL.</td>
<td>The API call expected a pointer to be supplied.</td>
<td>Check your application to see if the input to the API call is specified correctly.</td>
</tr>
<tr>
<td>DMB0162I</td>
<td>You set requestedHits to zero.</td>
<td>With requestedHits set to zero, you get nothing back.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0163I</td>
<td>That function is not yet supported.</td>
<td>That function is not yet supported.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0164E</td>
<td>The system cannot process the query (&lt;query-name&gt;).</td>
<td>An error occurred when the query was created.</td>
<td>Check the input to the command or API and try again.</td>
</tr>
<tr>
<td>DMB0165E</td>
<td>The system cannot run the query (&lt;query-name&gt;).</td>
<td>An error occurred when the query was created.</td>
<td>Check the input to the command or API and try again.</td>
</tr>
<tr>
<td>DMB0166E</td>
<td>A statement error was found in &quot;&lt;name&gt;&quot; while executing &quot;&lt;name&gt;&quot;: &quot;&lt;error&gt;&quot;.</td>
<td>An internal IBM error occurred.</td>
<td>Please contact your database administrator.</td>
</tr>
<tr>
<td>DMB0167E</td>
<td>An error occurred while reading QbGenericImageDataClass (&lt;error&gt;).</td>
<td>An internal IBM error occurred.</td>
<td>Please contact your database administrator.</td>
</tr>
<tr>
<td>DMB0168E</td>
<td>A query's feature &quot;&lt;feature-name&gt;&quot; was not set prior to search.</td>
<td>The query did not run because it had no feature assigned to it.</td>
<td>Add a feature to the query using either the QbAddFeature API or the ADD QBIC FEATURE command.</td>
</tr>
<tr>
<td>DMB0169E</td>
<td>The following error occurred in the Call-Level Interface: &quot;&lt;error&gt;&quot;.</td>
<td>CLI error.</td>
<td>Follow the directions in the message text.</td>
</tr>
<tr>
<td>DMB0170E</td>
<td>Query name &quot;&lt;query-name&gt;&quot; is already in use.</td>
<td>Another query exists with that name.</td>
<td>Select another name.</td>
</tr>
<tr>
<td>DMB0171E</td>
<td>Query name &quot;&lt;query-name&gt;&quot; has not been stored.</td>
<td>After creating the query, the system could not store it.</td>
<td>Make sure that you have write authority and</td>
</tr>
</tbody>
</table>
Messages

enough storage to store the query.

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Description</th>
<th>Cause</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMB0172E</td>
<td>SQL Error: &quot;&lt;error&gt;&quot;.</td>
<td>An internal error has occurred.</td>
<td>Follow the instructions that are contained in the associated error message and repeat the operation. If the error occurs again, contact IBM service.</td>
</tr>
<tr>
<td>DMB0173E</td>
<td>The catalog is open, but for read-only: &quot;&lt;catalog-name&gt;&quot;.</td>
<td>You cannot update the catalog because someone else already opened the catalog in write mode, or you do not have write authority for it.</td>
<td>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.</td>
</tr>
<tr>
<td>DMB0174E</td>
<td>A system error occurred: &quot;&lt;error&gt;&quot;.</td>
<td>An internal IBM error occurred.</td>
<td>Follow the instructions that are contained in the associated error message and repeat the operation. If the error occurs again, contact IBM service.</td>
</tr>
<tr>
<td>DMB0175I</td>
<td>Images were not found: &quot;&lt;information&gt;&quot;.</td>
<td>No images were found that matched the query. The database might be empty.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0176I</td>
<td>The column already has a QBIC catalog: &quot;&lt;table-name column-name&gt;&quot;.</td>
<td>Another catalog exists with that name.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0177E</td>
<td>The system cannot open the catalog; the error message is: &quot;&lt;error&gt;&quot;.</td>
<td>The catalog was damaged.</td>
<td>Follow the instructions in the message text.</td>
</tr>
<tr>
<td>DMB0178E</td>
<td>The system cannot delete the catalog; the error message is: &quot;&lt;error&gt;&quot;.</td>
<td>Either the catalog does not exist, or it was damaged.</td>
<td>Verify the name and existence of the catalog and try again.</td>
</tr>
<tr>
<td>DMB0179E</td>
<td>The catalog handle is not valid: &quot;&lt;error&gt;&quot;.</td>
<td>The catalog handle you used in the API call is not valid.</td>
<td>Check your application to see if you are obtaining the correct catalog handle.</td>
</tr>
<tr>
<td>DMB0180I</td>
<td>Access to catalog is denied: &quot;&lt;error&gt;&quot;.</td>
<td>Access is denied.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0181I</td>
<td>Catalog is in use &quot;&lt;error&gt;&quot;.</td>
<td>Another operation is using this catalog.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0184I</td>
<td>Tracing has already been started:</td>
<td></td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0185I</td>
<td>Tracing has not been started yet:</td>
<td></td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0186I</td>
<td>Tracing was turned on at &quot;&lt;time&gt;&quot; from directory &quot;&lt;directory-name&gt;&quot;. The trace file is &quot;&lt;file-name&gt;&quot;. &quot;&lt;Count&gt; bytes of trace data have been written.</td>
<td></td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0187E</td>
<td>Communication cannot be established because the system cannot open file &quot;&lt;file-name&gt;&quot; for writing.</td>
<td>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.</td>
<td>Log with the user ID that owns the instance. Verify that the environment variables are set correctly.</td>
</tr>
<tr>
<td>DMB0188I</td>
<td>An error occurred when creating the trace daemon: &quot;&lt;error&gt;&quot;</td>
<td>An internal error has occurred.</td>
<td>Repeat the operation. If the error occurs again, contact IBM service.</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Cause</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMB0204E</td>
<td>The column is not enabled for DB2IMAGE: &quot;&lt;column-name&gt;&quot;.</td>
<td>The column is not enabled for the Image Extender.</td>
<td>Make sure that the column is enabled for the DB2 Image Extender.</td>
</tr>
<tr>
<td>DMB0205E</td>
<td>No catalog found for &quot;&lt;table-name column-name&gt;&quot;.</td>
<td>There is no QBIC catalog associated with the specified column.</td>
<td>Create a QBIC catalog for the column before performing any other QBIC operations.</td>
</tr>
<tr>
<td>DMB0206W</td>
<td>The specified column is not enabled for any extender.</td>
<td>The column might not exist or its data type might not be compatible with the extenders.</td>
<td>Verify that the column has been defined using the correct data type.</td>
</tr>
<tr>
<td>DMB0207E</td>
<td>Can not overwrite the file.</td>
<td>The file already exists, but the EXPORT UDF cannot overwrite it.</td>
<td>Export the file to a different file name or allow the EXPORT UDF to overwrite the file.</td>
</tr>
<tr>
<td>DMB0208E</td>
<td>sqlcode=&lt;code&gt; clistate=&lt;code&gt;.</td>
<td>An internal error has occurred.</td>
<td>Repeat the operation. If the error occurs again, contact IBM service.</td>
</tr>
<tr>
<td>DMB0209E</td>
<td>Invalid audio header.</td>
<td>The header of the audio file is missing or corrupt.</td>
<td>Verify that the format of the audio file is supported by DB2 Extenders.</td>
</tr>
<tr>
<td>DMB0211W</td>
<td>File exists w/o overwrite.</td>
<td>The specified target file already exists and is not overwritten.</td>
<td>No action is required.</td>
</tr>
<tr>
<td>DMB0212E</td>
<td>The resultType parameter is reserved and must be 0.</td>
<td>The parameter is reserved for future use.</td>
<td>Set the resultType to 0.</td>
</tr>
<tr>
<td>DMB0214E</td>
<td>The pointer to the query name is not valid.</td>
<td>The API call expected a valid pointer to the input query name.</td>
<td>Make sure that valid parameters are supplied to the API call and the syntax is followed correctly.</td>
</tr>
<tr>
<td>DMB0352E</td>
<td>Command line environment not initialized.</td>
<td>The command line environment is not initialized to run the db2ext command-line processor.</td>
<td>Issue the db2cmd command to open a DB2CLP window, then issue the db2ext command to run the db2 command-line processor in that window.</td>
</tr>
<tr>
<td>DMB0353E</td>
<td>Cannot communicate with db2ext command-line processor’s background process.</td>
<td>The background process for the db2ext command-line processor is running, but the db2ext command-line processor cannot communicate with it.</td>
<td>Try issuing the db2ext command in a different window.</td>
</tr>
<tr>
<td>DMB0354E</td>
<td>Cannot start db2ext command-line processor’s background process.</td>
<td>The background process for the db2ext command-line processor is running, but the db2ext command-line processor cannot communicate with it.</td>
<td>Check that the executable module for the background process (db2extb or db2extb.exe) exists, and its directory is in the PATH environment variable.</td>
</tr>
<tr>
<td>DMB0355E</td>
<td>db2ext command-line processor’s background process timed out.</td>
<td>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.</td>
<td>Try issuing the db2ext command in a different window.</td>
</tr>
<tr>
<td>DMB0356E</td>
<td>Cannot communicate with the db2ext command-line processor’s background process.</td>
<td>The db2ext command-line processor sent a request to its background process, but the request was not received.</td>
<td>Make sure that the background process for...</td>
</tr>
</tbody>
</table>
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 db2ext 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-names>"."<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 colorcolor histogramfile.

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.
### Messages

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Message Description</th>
<th>Cause</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMB0374E</td>
<td>Expected percentage.</td>
<td>The percent value is expected in the query string.</td>
<td>Correct the query string and try again.</td>
</tr>
<tr>
<td>DMB0375E</td>
<td>Expected color.</td>
<td>The red, green, and blue values are expected in the query string.</td>
<td>Correct the query string and try again.</td>
</tr>
<tr>
<td>DMB0376E</td>
<td>Expected '='.</td>
<td>An '=' is expected in the query string.</td>
<td>Correct the query string and try again.</td>
</tr>
<tr>
<td>DMB0377E</td>
<td>Expected '&lt;'.</td>
<td>An '&lt;' is expected in the query string.</td>
<td>Correct the query string and try again.</td>
</tr>
<tr>
<td>DMB0378E</td>
<td>Expected '&gt;'.</td>
<td>An '&gt;' is expected in the query string.</td>
<td>Correct the query string and try again.</td>
</tr>
<tr>
<td>DMB0379E</td>
<td>Expected 'and'.</td>
<td>An 'and' is expected in the query string.</td>
<td>Correct the query string and try again.</td>
</tr>
<tr>
<td>DMB0380E</td>
<td>Expected weight.</td>
<td>A weight is expected in the query string.</td>
<td>Correct the query string and try again.</td>
</tr>
<tr>
<td>DMB0381E</td>
<td>Feature not set.</td>
<td>The feature has not been added to the QBIC catalog.</td>
<td>Add the feature to the QBIC catalog, and recatalog the images.</td>
</tr>
<tr>
<td>DMB0382E</td>
<td>Could not build query.</td>
<td>The extender server for the current database might be stopped.</td>
<td>Check the status of the server. If the server is not running, restart it using the START SERVER command on the db2ext command line.</td>
</tr>
<tr>
<td>DMB0383E</td>
<td>Could not execute query.</td>
<td>The extender server for the current database might be stopped.</td>
<td>Check the status of the server. If the server is not running, restart it using the START SERVER command on the db2ext command line.</td>
</tr>
<tr>
<td>DMB0384E</td>
<td>Could not get next item.</td>
<td>End of the list has been reached.</td>
<td>Make sure that your application is not attempting to retrieve an item beyond the end of the list.</td>
</tr>
<tr>
<td>DMB0390E</td>
<td>The command &quot;&lt;command-name&gt;&quot; or API &quot;&lt;api-name&gt;&quot; is valid only if the application is connected to a DB2 &quot;&lt;server-type&gt;&quot; server.</td>
<td>The application is connected to a type of DB2 server that the API or command does not support.</td>
<td>Use the proper API or command for the type of DB2 server to which the application is connected.</td>
</tr>
<tr>
<td>DMB0391I</td>
<td>This command can be run only when a DB2 UDB client is accessing a DB2 UDB server.</td>
<td>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.</td>
<td>Do not issue this command in the current client/server configuration.</td>
</tr>
<tr>
<td>DMB0392I</td>
<td>The command can be run only when a DB2 UDB client is accessing a DB2 UDB Extended Enterprise Edition server.</td>
<td>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.</td>
<td>Do not issue this command in the current client/server configuration.</td>
</tr>
</tbody>
</table>
DMB0395E  This command cannot be run from a DB2 for z/OS client.

Cause:  DB2 extender client code supports administrative APIs and commands for a DB2 for z/OS 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 z/OS.

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.

DMB0396I  The current server is disabled for the "<extender-name>" extender.

Cause:  The current server is disabled for the "<extender-name>" extender.

Action:  Do not issue this command in the current client/server configuration.

DMB0397E  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.

DMB0398E  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.

DMB0399E  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.

DMB0405E  -a option is only applicable if you are a DB2 z/OS client.

Cause:  The -a option is used to specify an z/OS subsystem, however it was issued from a DB2 UDB client.

Action:  Do not specify the -a option from a DB2 UDB client.

DMB0406E  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.

DMB0407E  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.

DMB0408E  Error on node "<node-number>".

Cause:  An error occurred on the specified node number.

Action:  Make the necessary corrections and try again.

DMB0409E  One or more nodes are not responding.

Cause:  One or more nodes are currently not responding.

Action:  Try again later.
Messages

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.

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 z/OS 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 path

...
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
z/OS" on page 414.

**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

z/OS Open Edition supports wildcard specification in file names. For example, the
following command in z/OS 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
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 245 lists and describes environment variables that you can set for use by the Image, Audio, and Video Extenders in resolving file names.

<table>
<thead>
<tr>
<th>Image Extender</th>
<th>Audio Extender</th>
<th>Video Extender</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2IMAGEPATH</td>
<td>DB2AUDIOPATH</td>
<td>DB2VIDEOPATH</td>
<td>Used to resolve source file name for store, retrieve, and update operations from a server file.</td>
</tr>
<tr>
<td>DB2IMAGESTORE</td>
<td>DB2AUDIOSTORE</td>
<td>DB2VIDEOSTORE</td>
<td>Used to resolve target file name for store and update operations to a server file.</td>
</tr>
<tr>
<td>DB2IMAGEEXPORT</td>
<td>DB2AUDIOEXPORT</td>
<td>DB2VIDEOEXPORT</td>
<td>Used to resolve target file name for retrieve operations to a server file.</td>
</tr>
<tr>
<td>DB2IMAGETEMP</td>
<td></td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image Extender</th>
<th>Audio Extender</th>
<th>Video Extender</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2IMAGEPATH</td>
<td>DB2AUDIOPATH</td>
<td>DB2VIDEOPATH</td>
<td>Used to resolve source file name for display and play operations on a client file.</td>
</tr>
<tr>
<td>DB2IMAGETEMP</td>
<td>DB2AUDIOTEMP</td>
<td>DB2VIDEOTEMP</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
</table>

© Copyright IBM Corp. 1998, 2001
Environment variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2MMPATH</td>
<td>Used to resolve source file name for store, retrieve, and update operations.</td>
</tr>
<tr>
<td>DB2MMSTORE</td>
<td>Used to resolve target file name for store and update operations.</td>
</tr>
<tr>
<td>DB2MMEXPORT</td>
<td>Used to resolve target file name for retrieve operations.</td>
</tr>
<tr>
<td>DB2MMTEMP</td>
<td>Used to resolve file name for operations that create temporary files.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2IMAGEBROWSER</td>
<td>Used to identify the default image display program.</td>
</tr>
<tr>
<td>DB2AUDIOPLAYER</td>
<td>Used to identify the default audio player program.</td>
</tr>
<tr>
<td>DB2VIDEOPLAYER</td>
<td>Used to identify the default video player program.</td>
</tr>
</tbody>
</table>

Setting environment variables

You can set environment variables in z/OS, AIX, Solaris, and Windows.

Setting environment variables in z/OS

In z/OS, environment variables are specified in a DSN in the DMBENVAR DD card in the WLM managed startup proc, for example, DMBWLML1. The following is a sample DMBWLML1 procedure:

```
//V61AWLM1 PROC DB2SSN=V61A,NUMTCB=18,APPLENV=DMBWLM1
//TCBNUM1 EXEC PGM=DSNX9WLM,TIME=1440,
//       PARM=’&DB2SSN,&NUMTCB,&APPLENV’,REGION=0M
//STELIB DD DSN=DSN610.SDSNL0,DISP=SHR <= DB2 Load Library
//       DSN=DMB.V61.L0LOAD,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
//+DSNADOTRC DD DISP=SHR,DSN=HQL.DMB.DSNADOTRC <= DB2 CLI Trace
```

The following is a sample of the contents of global environment variables defined in file DMB.ENVAR:

```
DB2MMPATH=/usr/lpp/db2ext_06_01_00/samples:/tmp
DB2MMTEMP=/tmp
DB2MMSTORE=/tmp
DB2MMEXPORT=/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**

```bash
setenv DB2MMPATH /tmp
setenv DB2MMTEMP /tmp
```

**Korn and Bourne shell**

```bash
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:

```bash
setenv env-var directory
```

For example:

```bash
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:

```bash
env-var=directory
export env-var
```

For example:

```bash
DB2MMPATH=/usr/lpp/db2ext/samples:/media
export DB2MMPATH

DB2IMAGEPATH=/employee/pictures:/images
export DB2IMAGEPATH

DB2AUDIOSTORE=/employee/sounds
export DB2AUDIOSTORE
```
Environment variables

Korn shell

Use the EXPORT command to set environment variables:

```bash
export env-var=directory
```

For example:

```bash
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.

<table>
<thead>
<tr>
<th>Open Edition</th>
<th>MVS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable.c</td>
<td>ENABLE</td>
<td>Enables a database server for the Audio, Image, and Video Extenders, creates a table, and enables the table and its columns.</td>
</tr>
<tr>
<td>populate.c</td>
<td>POPULATE</td>
<td>Imports data into the table.</td>
</tr>
<tr>
<td>query.c</td>
<td>QUERY</td>
<td>Queries the data in the table.</td>
</tr>
<tr>
<td>api.c</td>
<td>API</td>
<td>Queries the database server using extender APIs.</td>
</tr>
<tr>
<td>handle.c</td>
<td>HANDLE</td>
<td>Demonstrates the use of handles in UDFs and how to make where clause comparisons in SELECT statements.</td>
</tr>
<tr>
<td>qbcatdmo.c</td>
<td>QBCATDMO</td>
<td>Creates a QBIC catalog and catalogs a column of images into the catalog</td>
</tr>
<tr>
<td>qbidemo.c</td>
<td>QBICDEMO</td>
<td>Queries a QBIC catalog</td>
</tr>
<tr>
<td>color.c</td>
<td>COLOR</td>
<td>Makes color table declarations for qbidemo.c</td>
</tr>
<tr>
<td>utility.c</td>
<td>UTILITY</td>
<td>Utility routines.</td>
</tr>
<tr>
<td>utility.h</td>
<td>UTILITY</td>
<td>Header file for utility routines.</td>
</tr>
<tr>
<td>makefile.aix</td>
<td></td>
<td>Makefile to build the programs in AIX</td>
</tr>
<tr>
<td>makefile.iva</td>
<td></td>
<td>Makefile to build the programs in Windows NT (or later) using IBM VisualAge C++</td>
</tr>
<tr>
<td>makefile.mvc</td>
<td></td>
<td>Makefile to build the programs using Microsoft Visual C++</td>
</tr>
<tr>
<td>makefile.sun</td>
<td></td>
<td>Makefile to build the programs in Solaris</td>
</tr>
</tbody>
</table>

 Executable files are provided for the following sample programs. The sample programs are intended to be run in the order that is shown.
Sample programs

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 z/OS 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 419) 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 424). The default plan name is DMBACL.

   4. Set up the CLI.INI file (see Figure 27 on page 425).

   5. For TSO, set STEPLIB concatenation (see Figure 28 on page 425).

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=N,MSGLEVEL=(1,1),USER=&SYSUID,REGION=OK,
// NOTIFY=&SYSUID,PASSWORD=&SYSUID
//*ROUTE PRINT hostname.userid
//**************************************************************************
//DMBC JOB LNGPRFX='CBC',
// CLANG='EDCMSGE', USED IN THIS RELEASE. KEPT FOR COM
// CXXLANG='CBCMSGE', USED IN THIS RELEASE. KEPT FOR COM
// CREGSIZ='16M',
// CPARM=, < COMPILER OPTIONS
// CPARM2=, < COMPILER OPTIONS
// CPARM3=, < COMPILER OPTIONS
// LIBPRFX='SYS1',
// TUNIT='V1O',
// 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(UDEBUG,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(DBM_UNIX_DIRECTORY="/opt/IBMdb2ex/V6.0/")
NOMAR,NOSQ,LOCATE(POSIX),LANGLVL(ANSI)
OBJECT,LONGNAME,REN
DLL,EXPORTALL
NOAGGR,NOREF,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),
//MGCLASS=H,MSGLEVEL=(1,1),USER=&SYSUID,REGION=OK
//*/
//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
//SCEEOBJ DD DISP=SHR,DSN=&LIBPRFX..SCEEOBJ <== INPUT CEE/EC
//SCEEcpp DD DISP=SHR,DSN=&LIBPRFX..SCEEcpp <== INPUT CEE/EC
//SCLBCPP DD DISP=SHR,DSN=&LNGPRFX..SCLBCPP <== INPUT COMPIL
//SYSMSGS DD DISP=SHR,DSN=&LIBPRFX..SCEEMSGP(EDCPMSGE)
//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.DSNSMACS.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 SCEEOBJ
// LIBRARY SCEECPP

Figure 25. Sample JCL (Part 2 of 5)
Sample programs

//** 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(DMIMAGE)
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=OK
// * 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**
//SYSLMOD DD DISP=SHR,DSN=DMB.V61.SAMPLE.LOADLIB <= OUT LOAD MODS
//SYSLIN DD DUMMY
//DBMCLTLK PEND

Figure 25. Sample JCL (Part 4 of 5)
/*-------------------------------------------------------------------
/* COMMAND AND *CLT LINKEDIT LIST
/* INCLUDE DB2LIB(DSNTIAR)
/* INCLUDE DB2LIB(OSNLI)
/*-------------------------------------------------------------------

//DMBCLT EXEC DMBCCLT
//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)
Sample programs

```plaintext
//BINDALL  JOB CLASS=A,
// MSGCLASS=H,MSGLEVEL=(1,1),REGION=OK,TIME=NOLIMIT,
// USER=SYSADM,PASSWORD=SYSADM,NOTIFY=&SYSUID
//*ROUTE PRINT hostname.userid
//*****************************************************************************
//JOBLIB   DD DISP=SHR,DSN=DB2A.SDSNLOAD
//*/
// SET DMBFQP=DMB.V61A
//*/
//BINDCLI EXEC PGM=IKJEFT01,DYNAMNBR=20
//DBRMLIB DD DISP=SHR,
// DSN=REDEM.DBRMLIB.DATA
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSDUMP 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(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
```
Under TSO, a DD name of DSNAOINI must be allocated for the DB2 CLI initialization processing.

```bash
alloc dd(dsnaoini) da('HQL.cli.ini6') shr
```

Example of CLI.INI file(HQL.cli.ini6):

```ini
; 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

```bash
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
Sample media files
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
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
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 z/OS and develop programs for use with DB2 Extenders for z/OS. This publication documents General-use Programming Interface and Associated Guidance Information provided by DB2 Extenders for z/OS.

General-use programming interfaces allow you to write programs that obtain the services of DB2 Extenders for z/OS. You may copy the DB2 Extenders for z/OS 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 z/OS 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 z/OS V6.
The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

AIX
DB2
DB2 Extenders
DB2 Universal Database
IBM
MVS
OS/2
OS/390
QBIC
RACF
z/OS

Microsoft, Windows, Windows NT, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Intel is a registered trademark of Intel.

Other company, product, or service names may be trademarks or service marks of others.
Glossary

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. 

distinct type.  See user-defined type.

UDF.  See user-defined function.

UDT.  See user-defined type.
### Index

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>access privileges 32</td>
</tr>
<tr>
<td></td>
<td>ADD QBIC FEATURE command 111, 359</td>
</tr>
<tr>
<td></td>
<td>administration commands on client 357</td>
</tr>
<tr>
<td></td>
<td>ADD QBIC FEATURE 359</td>
</tr>
<tr>
<td></td>
<td>CATALOG QBIC COLUMN 360</td>
</tr>
<tr>
<td></td>
<td>CLOSE QBIC CATALOG 361</td>
</tr>
<tr>
<td></td>
<td>CREATE QBIC CATALOG 362</td>
</tr>
<tr>
<td></td>
<td>DELETE QBIC CATALOG 364</td>
</tr>
<tr>
<td></td>
<td>DISABLE COLUMN 365</td>
</tr>
<tr>
<td></td>
<td>DISABLE SERVER 366</td>
</tr>
<tr>
<td></td>
<td>DISABLE TABLE 367</td>
</tr>
<tr>
<td></td>
<td>ENABLE COLUMN 368</td>
</tr>
<tr>
<td></td>
<td>ENABLE SERVER 369</td>
</tr>
<tr>
<td></td>
<td>ENABLE TABLE 371</td>
</tr>
<tr>
<td></td>
<td>GET EXTENDER STATUS 373</td>
</tr>
<tr>
<td></td>
<td>GET INACCESSIBLE FILES 374</td>
</tr>
<tr>
<td></td>
<td>GET QBIC CATALOG INFO 376</td>
</tr>
<tr>
<td></td>
<td>GET REFERENCED FILES 377</td>
</tr>
<tr>
<td></td>
<td>GRANT 379</td>
</tr>
<tr>
<td></td>
<td>OPEN QBIC CATALOG 381</td>
</tr>
<tr>
<td></td>
<td>QUIT 382</td>
</tr>
<tr>
<td></td>
<td>REMOVE QBIC FEATURE 383</td>
</tr>
<tr>
<td></td>
<td>REVOKE 384</td>
</tr>
<tr>
<td></td>
<td>TERMINATE 386</td>
</tr>
<tr>
<td></td>
<td>administration task overview 37</td>
</tr>
<tr>
<td></td>
<td>administrative support tables 14</td>
</tr>
<tr>
<td></td>
<td>description 14</td>
</tr>
<tr>
<td></td>
<td>security 32</td>
</tr>
<tr>
<td></td>
<td>alignment value of audio or video 151</td>
</tr>
<tr>
<td></td>
<td>AlignValue UDF 151</td>
</tr>
<tr>
<td></td>
<td>allocation resources 69</td>
</tr>
<tr>
<td></td>
<td>application programming interfaces (APIs) 219</td>
</tr>
<tr>
<td></td>
<td>(continued)</td>
</tr>
<tr>
<td></td>
<td>DBiAdminGetInaccessibleFiles 252</td>
</tr>
<tr>
<td></td>
<td>DBiAdminGetReferencedFiles 254</td>
</tr>
<tr>
<td></td>
<td>DBiAdminIsFileReferenced 256</td>
</tr>
<tr>
<td></td>
<td>DBiBrowse 258</td>
</tr>
<tr>
<td></td>
<td>DBiDisableColumn 260</td>
</tr>
<tr>
<td></td>
<td>DBiDisableServer 262</td>
</tr>
<tr>
<td></td>
<td>application programming interfaces (APIs) (continued)</td>
</tr>
<tr>
<td></td>
<td>DBiDisableTable 263</td>
</tr>
<tr>
<td></td>
<td>DBiEnableColumn 265</td>
</tr>
<tr>
<td></td>
<td>DBiEnableServer 267</td>
</tr>
<tr>
<td></td>
<td>DBiEnableTable 269</td>
</tr>
<tr>
<td></td>
<td>DBiGetError 271</td>
</tr>
<tr>
<td></td>
<td>DBiGetInaccessibleFiles 272</td>
</tr>
<tr>
<td></td>
<td>DBiGetReferencedFiles 274</td>
</tr>
<tr>
<td></td>
<td>DBiIsColumnEnabled 276</td>
</tr>
<tr>
<td></td>
<td>DBiIsFileReferenced 278</td>
</tr>
<tr>
<td></td>
<td>DBiIsServerEnabled 280</td>
</tr>
<tr>
<td></td>
<td>DBiIsTableEnabled 281</td>
</tr>
<tr>
<td></td>
<td>DBiPrepareAttrs 283</td>
</tr>
<tr>
<td></td>
<td>DBvAdminGetInaccessibleFiles 284</td>
</tr>
<tr>
<td></td>
<td>DBvAdminGetReferencedFiles 286</td>
</tr>
<tr>
<td></td>
<td>DBvAdminIsFileReferenced 288</td>
</tr>
<tr>
<td></td>
<td>DBvDisableColumn 290</td>
</tr>
<tr>
<td></td>
<td>DBvDisableServer 292</td>
</tr>
<tr>
<td></td>
<td>DBvDisableTable 293</td>
</tr>
<tr>
<td></td>
<td>DBvEnableColumn 295</td>
</tr>
<tr>
<td></td>
<td>DBvEnableServer 297</td>
</tr>
<tr>
<td></td>
<td>DBvEnableTable 299</td>
</tr>
<tr>
<td></td>
<td>DBvGetError 301</td>
</tr>
<tr>
<td></td>
<td>DBvGetInaccessibleFiles 302</td>
</tr>
<tr>
<td></td>
<td>DBvGetReferencedFiles 304</td>
</tr>
<tr>
<td></td>
<td>DBvIsColumnEnabled 306</td>
</tr>
<tr>
<td></td>
<td>DBvIsFileReferenced 308</td>
</tr>
<tr>
<td></td>
<td>DBvIsServerEnabled 310</td>
</tr>
<tr>
<td></td>
<td>DBvIsTableEnabled 311</td>
</tr>
<tr>
<td></td>
<td>DBvPlay 313</td>
</tr>
<tr>
<td></td>
<td>DBvPrepareAttrs 315</td>
</tr>
<tr>
<td></td>
<td>QbAddFeature 316</td>
</tr>
<tr>
<td></td>
<td>QbCatalogColumn 318</td>
</tr>
<tr>
<td></td>
<td>QbCloseCatalog 320</td>
</tr>
<tr>
<td></td>
<td>QbCreateCatalog 321</td>
</tr>
<tr>
<td></td>
<td>QbDeleteCatalog 323</td>
</tr>
<tr>
<td></td>
<td>QbGetCatalogInfo 325</td>
</tr>
<tr>
<td></td>
<td>QbListFeatures 327</td>
</tr>
<tr>
<td></td>
<td>QbOpenCatalog 329</td>
</tr>
<tr>
<td></td>
<td>QbQueryAddFeature 331</td>
</tr>
<tr>
<td></td>
<td>QbQueryCreate 333</td>
</tr>
<tr>
<td></td>
<td>QbQueryDelete 334</td>
</tr>
<tr>
<td></td>
<td>QbQueryGetFeatureCount 335</td>
</tr>
<tr>
<td></td>
<td>QbQueryGetString 336</td>
</tr>
<tr>
<td></td>
<td>QbQueryListFeatures 337</td>
</tr>
<tr>
<td></td>
<td>QbQueryNameCreate 339</td>
</tr>
<tr>
<td></td>
<td>QbQueryNameDelete 341</td>
</tr>
<tr>
<td></td>
<td>QbQueryNameSearch 342</td>
</tr>
<tr>
<td></td>
<td>QbQueryRemoveFeature 344</td>
</tr>
<tr>
<td></td>
<td>QbQuerySearch 346</td>
</tr>
<tr>
<td></td>
<td>QbQuerySetFeatureData 348</td>
</tr>
<tr>
<td></td>
<td>QbQuerySetFeatureWeight 350</td>
</tr>
<tr>
<td></td>
<td>QbQueryStringLength 351</td>
</tr>
<tr>
<td></td>
<td>QbReCatalogColumn 353</td>
</tr>
<tr>
<td></td>
<td>QbRemoveFeature 355</td>
</tr>
<tr>
<td></td>
<td>aspect ratio of video 152</td>
</tr>
<tr>
<td></td>
<td>AspectRatio UDF 152</td>
</tr>
<tr>
<td></td>
<td>attributes, object 88</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1998, 2001
attributes, object (continued)
  alignment value 151
  aspect ratio 152
  audio channels (number of) 192
  bits per sample of audio 153
  clock speed per quarter note 213
  clock speed per second 214
  colors in image (number of) 193
  comment 155
  compression format of video 157
  data transfer rate of audio 154
  data transfer rate of video 190
  description 88
  duration of audio or video 179
  file name 180
  format 183
  frame rate of video 184
  frames in video (number of) 194
  height 187
  import time 189
  importer 188
  number of audio channels 192
  number of audio tracks 191
  number of colors in image 193
  number of frames in video 194
  number of video tracks 195
  playing time of audio or video 179
  sampling rate of audio 209
  size 210
  throughput of audio 154
  throughput of video 184, 190
  time stored 189
  time updated 216
  track name, MIDI 182
  track names, MIDI 186
  track number of all MIDI instruments 185
  track number of MIDI instrument 181
  update time 216
  updater 215
  user ID of person who stored 188
  user ID of person who updated 215
  video tracks (number of) 195
  width 217

audio
  alignment 151
  bits per sample 153
  channels (number of) 192
  clock speed, MIDI 213, 214
  comment attribute 155
  data transfer rate 154
  duration 179
  file name 180
  format attribute 183
  formats 71
  identifying format for storage 79
  identifying format for update 98
  import time 189
  importer 188
  number of channels 192
  number of tracks 191
  playing 103

audio (continued)
  playing time 179
  retrieving 84
  sampling rate 209
  size 210
  storing 73
  throughput 154
  time stored 189
  time updated 216
  track name, MIDI 182
  track names, MIDI 186
  track number of all MIDI instruments 185
  track number of MIDI instrument 181
  tracks in (number of) 191
  update time 216
  updater 215
  updating 90
  user ID of person who stored 188
  user ID of person who updated 215

Audio Extender 4
  DBaAdminGetInaccessibleFiles API 220
  DBaAdminGetReferencedFiles API 222
  DBaAdminIsFileReferenced API 224
  DBaDisableColumn API 226
  DBaDisableServer API 228
  DBaDisableTable API 229
  DBaEnableColumn API 231
  DBaEnableServer API 233
  DBaEnableTable API 235
  DBaGetConversion API 237
  DBaGetInaccessibleFiles API 238
  DBaGetReferencedFiles API 240
  DBaColumnEnabled API 242
  DBaColumnReferenced API 244
  DBaServerEnabled API 246
  DBaServerReferenced API 247
  DBaPlay API 249
  overview 4
  UDFs 147
  UDTs 147
  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 153
  buffer, client 64
    retrieving to with conversion 86
    retrieving to without format conversion 85
    storing from 77
    transmitting an object to or from 64
buffer, client (continued)
  updating from 96
BytesPerSec UDF 154

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 113, 360
  cataloging a column 113
  recataloging an image 114
channels, number of audio 192
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, 361
coarseness 17
codes, return 387
colors, number of (in image) 193
columns 48
  disabling 49
  enabling 48
commands 357
  ADD QBIC FEATURE 359
  CATALOG QBIC COLUMN 360
  CLOSE QBIC CATALOG 361
  CREATE QBIC CATALOG 362
  DELETE QBIC CATALOG 364
  DISABLE COLUMN 365
  DISABLE SERVER 366
  DISABLE TABLE 367
  ENABLE COLUMN 368
  ENABLE SERVER 369
  ENABLE TABLE 371
  GET EXTENDER STATUS 373
  GET INACCESSIBLE FILES 374
  GET QBIC CATALOG INFO 376
  GET REFERENCED FILES 377
  GRANT 379
commands (continued)
  OPEN QBIC CATALOG 381
  QUIT 382
  REMOVE QBIC FEATURE 383
  REVOKE 384
  TERMINATE 386
comment 83
  retrieving 89
  storing 83
  updating 101
Comment UDF 155
compatibility UDF 158
compiling 67
compression format of video 157
compression type 72
CompressType UDF 157
concepts 11
Content UDF 158
ContentA UDF 162
contrast 17
conversion options, image 72
CREATE QBIC CATALOG command 108, 362
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 154
data transfer rate of video 190
Database Request Module (DBRM) 67
database servers 42
  checking if enabled 51
  enabling 42
DB2 extender 3
codes 387, 388
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 387
run-time environment 4
sample media files 417
sample programs 417
scenario 19
security 32
Software Developers Kit (SDK) 4
SQLSTATE codes 388
storing objects using 71
tasks that can be performed with 60
trace facility 410
UDFs 147
UDTs 147
environment variables  (continued)
DB2VIDEOTEMP  413
external security  41

<table>
<thead>
<tr>
<th>F</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>features, QBIC query  122</td>
<td></td>
</tr>
<tr>
<td>file  64</td>
<td></td>
</tr>
<tr>
<td>finding files referenced by tables  53</td>
<td></td>
</tr>
<tr>
<td>name (that contains object)  180</td>
<td></td>
</tr>
<tr>
<td>names, relative  66</td>
<td></td>
</tr>
<tr>
<td>names, specifying  66</td>
<td></td>
</tr>
<tr>
<td>storing from client  77</td>
<td></td>
</tr>
<tr>
<td>transmitting an object between a table and  64</td>
<td></td>
</tr>
<tr>
<td>transmitting an object to or from client  65</td>
<td></td>
</tr>
<tr>
<td>updating from client  96</td>
<td></td>
</tr>
<tr>
<td>file reference variable  65</td>
<td></td>
</tr>
<tr>
<td>Filename UDF  180</td>
<td></td>
</tr>
<tr>
<td>FindInstrument UDF  181</td>
<td></td>
</tr>
<tr>
<td>FindTrackName UDF  182</td>
<td></td>
</tr>
<tr>
<td>Format UDF  183</td>
<td></td>
</tr>
<tr>
<td>formats of objects  71</td>
<td></td>
</tr>
<tr>
<td>handled by DB2 Extenders  71</td>
<td></td>
</tr>
<tr>
<td>identifying for storage  79</td>
<td></td>
</tr>
<tr>
<td>identifying for update  98</td>
<td></td>
</tr>
<tr>
<td>retrieving video  157</td>
<td></td>
</tr>
<tr>
<td>using your own for storage  81</td>
<td></td>
</tr>
<tr>
<td>using your own for update  99</td>
<td></td>
</tr>
<tr>
<td>FrameRate UDF  184</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>G</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GET EXTENDER STATUS command  373</td>
<td></td>
</tr>
<tr>
<td>GET INACCESSIBLE FILES command  374</td>
<td></td>
</tr>
<tr>
<td>GET QBIC CATALOG INFO command  112, 376</td>
<td></td>
</tr>
<tr>
<td>GET REFERENCED FILES command  377</td>
<td></td>
</tr>
<tr>
<td>GetInstruments UDF  185</td>
<td></td>
</tr>
<tr>
<td>GetTrackNames UDF  186</td>
<td></td>
</tr>
<tr>
<td>goal mode  32</td>
<td></td>
</tr>
<tr>
<td>GRANT command  379</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>handle  15</td>
<td></td>
</tr>
<tr>
<td>header files  63</td>
<td></td>
</tr>
<tr>
<td>Height UDF  187</td>
<td></td>
</tr>
<tr>
<td>hierarchical file system (HFS)  12</td>
<td></td>
</tr>
<tr>
<td>histogram color  16</td>
<td></td>
</tr>
<tr>
<td>description  16</td>
<td></td>
</tr>
<tr>
<td>feature name  111</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image  3</td>
<td></td>
</tr>
<tr>
<td>average color  16</td>
<td></td>
</tr>
<tr>
<td>colors in (number of)  193</td>
<td></td>
</tr>
<tr>
<td>comment attribute  155</td>
<td></td>
</tr>
<tr>
<td>compression type  72</td>
<td></td>
</tr>
<tr>
<td>conversion options  72</td>
<td></td>
</tr>
<tr>
<td>displaying  103</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>image (continued)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>file name  180</td>
<td></td>
</tr>
<tr>
<td>format attribute  183</td>
<td></td>
</tr>
<tr>
<td>height  187</td>
<td></td>
</tr>
<tr>
<td>height conversion  72</td>
<td></td>
</tr>
<tr>
<td>histogram color  16</td>
<td></td>
</tr>
<tr>
<td>identifying format for storage  79</td>
<td></td>
</tr>
<tr>
<td>identifying format for update  98</td>
<td></td>
</tr>
<tr>
<td>import time  189</td>
<td></td>
</tr>
<tr>
<td>importer  188</td>
<td></td>
</tr>
<tr>
<td>number of colors in  193</td>
<td></td>
</tr>
<tr>
<td>pixel  16</td>
<td></td>
</tr>
<tr>
<td>positional color  17</td>
<td></td>
</tr>
<tr>
<td>query by content  107</td>
<td></td>
</tr>
<tr>
<td>retrieving  84</td>
<td></td>
</tr>
<tr>
<td>rotation  72</td>
<td></td>
</tr>
<tr>
<td>score (QBIC)  132</td>
<td></td>
</tr>
<tr>
<td>size  210</td>
<td></td>
</tr>
<tr>
<td>storing  73</td>
<td></td>
</tr>
<tr>
<td>texture  17</td>
<td></td>
</tr>
<tr>
<td>time stored  189</td>
<td></td>
</tr>
<tr>
<td>time updated  216</td>
<td></td>
</tr>
<tr>
<td>update time  216</td>
<td></td>
</tr>
<tr>
<td>updater  215</td>
<td></td>
</tr>
<tr>
<td>updating  90</td>
<td></td>
</tr>
<tr>
<td>user ID of person who stored  188</td>
<td></td>
</tr>
<tr>
<td>user ID of person who updated  215</td>
<td></td>
</tr>
<tr>
<td>width  217</td>
<td></td>
</tr>
<tr>
<td>width conversion  72</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image Extender  4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DBaPrepareAttrs API  251</td>
<td></td>
</tr>
<tr>
<td>DBiAdminGetInaccessibleFiles API  252</td>
<td></td>
</tr>
<tr>
<td>DBiAdminGetReferencedFiles API  254</td>
<td></td>
</tr>
<tr>
<td>DBiAdminIsFileReferenced API  256</td>
<td></td>
</tr>
<tr>
<td>DBiBrowse API  258</td>
<td></td>
</tr>
<tr>
<td>DBiDisableColumn API  260</td>
<td></td>
</tr>
<tr>
<td>DBiDisableServer API  262</td>
<td></td>
</tr>
<tr>
<td>DBiDisableTable API  263</td>
<td></td>
</tr>
<tr>
<td>DBiEnableColumn API  265</td>
<td></td>
</tr>
<tr>
<td>DBiEnableServer API  267</td>
<td></td>
</tr>
<tr>
<td>DBiEnableTable API  269</td>
<td></td>
</tr>
<tr>
<td>DBiGetError API  271</td>
<td></td>
</tr>
<tr>
<td>DBiGetInaccessibleFiles API  272</td>
<td></td>
</tr>
<tr>
<td>DBiGetReferencedFiles API  274</td>
<td></td>
</tr>
<tr>
<td>DBilsColumnEnabled API  276</td>
<td></td>
</tr>
<tr>
<td>DBilsFileReferenced API  278</td>
<td></td>
</tr>
<tr>
<td>DBilsServerEnabled API  280</td>
<td></td>
</tr>
<tr>
<td>DBilsTableEnabled API  281</td>
<td></td>
</tr>
<tr>
<td>DBiPrepareAttrs API  283</td>
<td></td>
</tr>
<tr>
<td>DBvPrepareAttrs API  315</td>
<td></td>
</tr>
<tr>
<td>overview  4</td>
<td></td>
</tr>
<tr>
<td>UDFs  147</td>
<td></td>
</tr>
<tr>
<td>UDTs  147</td>
<td></td>
</tr>
<tr>
<td>Importer UDF  188</td>
<td></td>
</tr>
<tr>
<td>ImportTime UDF  189</td>
<td></td>
</tr>
<tr>
<td>include files  63</td>
<td></td>
</tr>
<tr>
<td>description  63</td>
<td></td>
</tr>
</tbody>
</table>
J
job DMBSETUP, editing and running  124

L
large object (LOB)  11
description  11
displaying  103
playing  103
transmitting  64
link editing  68
LOB (large object)  11
description  11
displaying  103
locator  65
playing  103
transmitting  64
locator  65

M
MaxBytesPerSec UDF  190
media files  417
metadata tables  14
description  14
security  32
MIDI instrument  185
MMDB_STORAGE_TYPE_EXTERNAL  79
duration of audio or video  179
file name  180
format  183
formats  71
frame rate of video  184
frames in video (number of)  194
height  187
import time  189
importer  188
number of audio channels  192
number of audio tracks  191
number of colors in image  193
number of frames in video  194
number of video tracks  195
playing  103
playing time of audio or video  179
retrieving  84
sampling rate of audio  209
security  32
size  210
storing  73
throughput of audio  154
throughput of video  184, 190
thumbnail  211
time stored  189
time updated  216
transmitting  64
update time  216
updater  215
updating  90
user ID of person who stored  188
user ID of person who updated  215
video tracks (number of)  195
width  217
object orientation  11
ODBC dynamic link library  68
ODBC environment  67
OPEN QBIC CATALOG command  110, 381
operating environments for DB2 Extenders  9
overloaded function names  13
overview of DB2 Extenders  3
overwrite indicator  86

N
Notices  427
NumAudioTracks UDF  191
number of bits to represent image  72
NumChannels UDF  192
NumColors UDF  193
NumFrames UDF  194
NumVideoTracks UDF  195

O
object  11
alignment of  151
aspect ratio of  152
attributes, retrieving  88
audio channels (number of)  192
audio tracks (number of)  191
bits per sample of audio  153
colors in image (number of)  193
comment  155
compression format of video  157
data transfer rate of audio  154
data transfer rate of video  190
description  11
displaying  103
object (continued)
duration of audio or video  179
file name  180
format  183
formats  71
frame rate of video  184
frames in video (number of)  194
height  187
import time  189
importer  188
number of audio channels  192
number of audio tracks  191
number of colors in image  193
number of frames in video  194
number of video tracks  195
playing  103
playing time of audio or video  179
retrieving  84
sampling rate of audio  209
security  32
size  210
storing  73
throughput of audio  154
throughput of video  184, 190
thumbnail  211
time stored  189
time updated  216
transmitting  64
update time  216
updater  215
updating  90
user ID of person who stored  188
user ID of person who updated  215
video tracks (number of)  195
width  217
object orientation  11
ODBC dynamic link library  68
ODBC environment  67
OPEN QBIC CATALOG command  110, 381
operating environments for DB2 Extenders  9
overloaded function names  13
overview of DB2 Extenders  3
overwrite indicator  86

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  179
positional color  17
description  17
feature name  111
precompiling  67
prelinking  68
preparing a DB2 extender application  67

438  Image, Audio, and Video Extenders Administration and Programming
QbTextureFeatureClass 111
QbScoreTBFromStr 326
QbScoreTBFromName 327
QbScoreFromStr 328
QbScoreFromName 329
QbRemoveFeature 330
QbReCatalogColumn 331
QbQueryStringSearch 332
QbQuerySetFeatureWeight 333
QbQuerySetFeatureData 334
QbQuerySearch 335
QbQueryRemoveFeature 336
QbQueryNameSearch 337
QbQueryNameDelete 338
QbQueryNameCreate 339
QbQueryListFeatures 340
QbQueryGetString 341
QbQueryGetFeatureCount 342
QbQueryDelete 343
QbQueryCreate 344
QbQueryAddFeature 345
QbOpenCatalog 346
QbListFeatures 347
QbImageSource 348
QbHistogramColor 349
QbGetCatalogInfo 350
QbDrawFeatureClass 351
QbDeleteCatalog 352
QbCreateCatalog 353
QbCloseCatalog 354
QbCatalogColumn 355
QbAddFeature 356

Query by Image Content (QBIC) (continued)
QbQueryCreate API 333
QbQueryDelete API 334
QbQueryGetFeatureCount API 335
QbQueryGetString API 336
QbQueryListFeatures API 337
QbQueryNameCreate API 338
QbQueryNameDelete API 341
QbQueryNameSearch API 342
QbQueryRemoveFeature API 344
QbQuerySearch API 346
QbQuerySetFeatureData API 348
QbQuerySetFeatureWeight API 350
QbQueryStringSearch API 351
QbReCatalogColumn API 353
QbRemoveFeature API 355
steps 107
query string, QBIC 122
reusing 128
query, QBIC 121
building 121
issuing 130
QUIT command 382

R
recovery 35
reference variable, file 65
relative file names 66
REMOVE QBIC FEATURE command 112, 383
Replace UDF 203
ReplaceA UDF 206
retrieving an object 84
return codes 387
return codes (SQLSTATE) 388
REVOKE command 384
rotation of image 72
run-time environment 4

S
sample media files 417
sample programs 417
to
sampling rate of audio 209
SamplingRate UDF 209
scaling factor 72
schema name 13
score, image (QBIC) 132
security 32
segment 65
server file 64
retrieving to 86
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 210
Size UDF 210
Software Developers Kit (SDK) 4
SQLSTATE codes 388
storing an object 73
string, QBIC query 122

T

tables 45
disabling 49
enabling 45
TERMINATE command 386
Text Extender 4
texture 17
description 17
feature name 111
throughput of audio 154
throughput of video 190
thumbnail 82
displaying 105
storing 82
updating 100
Thumbnail UDF 211
TicksPerQNote UDF 213
TicksPerSec UDF 214
trace facility 410
track names, MIDI 186
track number of MIDI instrument 181
track number, MIDI 182
tracks 191
number of audio 191
number of video 195
transmitting large objects 64
trigger 14

U

UDF (user-defined function) 12
AlignValue 151
AspectRatio 152
BitsPerSample 153
BytesPerSec 154
Comment 155
CompressType 157
Content 158
ContentA 162
current path 13
DB2Audio 164
DB2AudioA 166
DB2Image 169
DB2ImageA 172
DB2Video 174
DB2VideoA 176
description 12
Duration 179
Filename 180
FindInstrument 181
FindTrackName 182
Format 183
FrameRate 184
GetInstruments 185

UDF (user-defined function) (continued)
GetTrackNames 186
Height 187
Importer 188
ImportTime 189
MaxBytesPerSec 190
names 13
NumAudioTracks 191
NumChannels 192
NumColors 193
NumFrames 194
NumVideoTracks 195
overloaded 13
QbScoreFromName 196
QbScoreFromStr 198
QbScoreTBFFromName 199
QbScoreTBFFromStr 201
reference 147
Replace 203
ReplaceA 206
SamplingRate 209
signature 14
Size 210
Thumbnum 211
TicksPerQNote 213
TicksPerSec 214
Updater 215
UpdateTime 216
Width 217
UDT (user-defined type) 12
description 12
names 13
Unicode support 70
Updater UDF 215
UpdateTime UDF 216
updating an object 90
user-defined function 12
AlignValue 151
AspectRatio 152
BitsPerSample 153
BytesPerSec 154
Comment 155
CompressType 157
Content 158
ContentA 162
current path 13
DB2Audio 164
DB2AudioA 166
DB2Image 169
DB2ImageA 172
DB2Video 174
DB2VideoA 176
description 12
Duration 179
Filename 180
FindInstrument 181
FindTrackName 182
Format 183
FrameRate 184
GetInstruments 185
GetTrackNames 186

440 Image, Audio, and Video Extenders Administration and Programming
video (continued)
  storing 73
  throughput (bytes per second) 190
  throughput (frame rate) 194
  thumbnail 211
  time stored 189
  time updated 216
  update time 216
  updater 215
  updating 90
  user ID of person who stored 188
  user ID of person who updated 215
  video tracks in (number of) 195
  width 217

Video Extender 4
  DBvAdminGetInaccessibleFiles API 284
  DBvAdminGetReferencedFiles API 286
  DBvAdminIsFileReferenced API 288
  DBvDisableColumn API 290
  DBvDisableServer API 292
  DBvDisableTable API 293
  DBvEnableColumn API 295
  DBvEnableServer API 297
  DBvEnableTable API 299
  DBvGetError API 301
  DBvGetInaccessibleFiles API 302
  DBvGetReferencedFiles API 304
  DBvIsColumnEnabled API 306
  DBvIsFileReferenced API 308
  DBvIsServerEnabled API 310
  DBvIsTableEnabled API 311
  DBvPlay API 313
  overview 4
  UDFs 147
  UDTs 147

W
  wait indicator 105
  width of object 217
  Width UDF 217
  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. SC26-9947-01

Overall, how satisfied are you with the information in this book?

<table>
<thead>
<tr>
<th>Overall satisfaction</th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How satisfied are you that the information in this book is:

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accurate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to find</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to understand</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Well organized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
Readers’ Comments — We’d Like to Hear from You

IBM Deutschland Entwicklung GmbH
Information Development, Dept 0446
Schoenaicher Strasse 220
71032 Boeblingen
Germany
IBM

Program Number: 5675-DB2